[SQUEAKING] [RUSTLING] [CLICKING] All right, so today's lecture.
Introduction to neural networks and deep learning.
So we'll start with a very quick intro to these things, and then we'll switch and dive deep into neural networks.
All right, so the field of AI originated in 1956.
Sadly, it didn't originate at MIT.
It originated at Dartmouth because all these people got together at Dartmouth-- I guess it's got a nice quad or whatever-- they got together, they defined the field.
But fortunately for us, MIT was very well represented.
So we have Marvin Minsky, who founded the MIT AI Lab-- John McCarthy, who invented LISP and then later defected to the West Coast-- and then Claude Shannon, who invented information theory, who was a professor at MIT.
So MIT was well represented.
These folks founded the field.
And they were so bright, they thought that AI was going to be substantially solved, quote unquote, "by that fall." Now, obviously, it turned out a bit differently than what they expected.
So it's been, whatever, 67, 68 years since its founding.
So it's gone through, essentially, in my opinion, three seminal breakthroughs, starting with the traditional approach, then machine learning, deep learning, and generative AI.
So let's take a very quick look at each of these breakthroughs and what motivated them.
So let's start with the traditional approach to AI.
And so what is AI?
AI, informally, is the ability to imbue computers with the ability to do things that only humans can typically do-- cognitive tasks, thinking tasks, and things like that.
And so the most common, sensical way to do that is to say, well, if I want the computer to do something complicated, like play chess, I'm just going to sit down with a few chess grandmasters, show them a whole bunch of board moves, and ask them how they figure out how to respond, how to play the next move.
I'm going to sit down and talk to all these people, and then I'm going to write down a whole bunch of rules.
If this is the board position, move this.
If this is the board position, move this, and so on and so forth.
Or I might sit down with a cardiologist and tell them, OK, how do you actually interpret an ECG?
They will give me all this-- similarly, a bunch of if-then rules.
I will take all these rules, I'll put them into the computer, and boom, I have a system that can do what a human can do.
Now, this approach, even though it's commonsensical, it kind of makes sense.
It had success in only a few areas.
And so the interesting question is, why was it not pervasively successful?
Why was it not pervasively successful?
It seems like a pretty good idea to me.
And the people who came up with these things are smart people.
They're not dumb people.
They know what they're doing.
So why did it not work?
AUDIENCE: Because it's time-intensive, so in the case that you have to run through all these scenarios that can ever exist, and still some new scenarios can come up that you didn't cater for [INAUDIBLE.] RAMA RAMAKRISHNAN: Right.
So there are two aspects to what you said, which is the first aspect is it's time-intensive.
That, as it turns out, is not a big deal because computers are getting faster and faster.
The second thing is actually the key thing, which is that it doesn't generalize to new situations very well.

The problem is there are an infinite number of things that you're going to see when you deploy these systems in the real world.
By definition, what you're training it on is a small sample of rules.
So these rules are very brittle.
But there's actually an even more interesting reason, and that reason is that we know more than we can tell.
This is called "Polanyi's paradox." So the idea is that if I come to you and say, hey, here's a picture, is it a dog or a cat?
You will tell me within-- I believe they measured it, like, 20 milliseconds or something.
You know if it's a dog or a cat.
And then if I ask you to explain to me exactly how you figured that out, you'll come up with a bunch of reasons-- alleged reasons.
Oh, if it has whiskers, I think it's a cat, or whatever.
But the problem is that you actually, first of all, can't really articulate what's going on in your head, how you do these things.
And number 2, even if you articulate it, oftentimes your articulation has no correspondence with how your brain actually does it.
So you're incomplete underlier.
So this is Polanyi's paradox.
So if you can't even tell me how you do something, how the heck am I supposed to take it and put it into a computer?
Doesn't work.
And second is the fact that we can't write down these rules for all possible situations-- edge cases, corner cases, cetera-- and the world is full of edge cases.
So for these reasons, this approach didn't work.
And so a different approach was developed, and this approach was, well, basically said, hey, instead of explicitly telling the computer what to do, why don't we simply give it lots of examples of inputs and outputs?
Chess positions-- next move.
ECG-- diagnosis-- inputs and outputs.
And then why don't we just use some statistical techniques to learn a mapping a function that can go from the input to the output?
That was the idea, and this idea is machine learning.
So machine learning is basically just a fancy way of saying learn from input-output examples using statistical techniques.
Good.
All right, so now there are numerous ways to create machine learning models.
And if you have ever done linear regression, congratulations, you've been doing machine learning.
And only one of those methods happens to be something called "neural networks." There are many other methods, and, in fact, you probably have done these other methods if you have done a course like The Analytics Edge or something similar.
So machine learning has got tremendous impact around the world.
It's like, at this point, it's widely accepted.
It's a very, very successful technology.
And in fact, whenever people are actually talking about AI, chances are they're actually talking about machine learning.
It's just that AI sounds cooler.
The only problem is for machine learning to work really well, the input data has to be structured.
And what I mean by that is data that can essentially be numericalized and stuck into the columns and rows of a spreadsheet.
So for example, here, let's say I want to put together a dataset of patients, their symptoms,

and their characteristics, and then in the following year, after they showed up at the doctor's office, whether they had a cardiac event or not.
I might create a dataset like this with age, smoking status, yes-no exercise, blah, blah, blah, blah, blah.
And so either these numbers are numbers-- they're numerical-- or if they are not numerical, they're categorical.
Yes-no smoking, yes-no-- things like that.
Which means that if you have categorical variables, you can just numericalize them pretty easily.
You folks have done some machine learning before, so you know things like one-hot encoding and stuff like that can be done to make them all numerical.
So the point is, you can just render the data into the columns and rows of a spreadsheet pretty easily.
That's what I mean by structured data.
But the situation is very different if you have unstructured data.
So if you have an image of a cute puppy-- this is my puppy, by the way, from many years ago.
Sadly, he's no more, but his name was Google.
[LAUGHTER] So, yeah, anyway, my DMD alums know Google well.
So this Google.
If you want to take Google-- this picture-- and figure out how to numericalize it, the first thing you want to-- need to understand is that if you actually look at how this picture is represented digitally in the computer, basically every picture like this is depicted using three tables of numbers.
And we'll get to what these numbers mean later on, but the point I'm making is that each number basically represents the amount of light-- on a scale of 0 to 255, the amount of light in that location, in that pixel.
That's all-- the amount of light.
So basically, this table is the amount of-- sorry-- this table is amount of red light, amount of green light, amount of blue light.
Now, you will agree with me that if you, for example, look at something like this and say, OK, 251 at this location-- there is a lot of blue light because it's 251 out of a possible 255.
Maybe a lot of blue light somewhere here.
There's a lot of blue here.
Whether that area is blue because of a piece of sky, some water, or a bunch of blue paint-- could be anything-- it's going to say 251.
So the underlying reality, the underlying object that's being described, has nothing to do with the 251, so that's the whole problem.
The raw form of the data has no intrinsic meaning with the underlying thing.
So given that there's no connection between the number and what it's describing, how the heck can any algorithm do anything with it?
It can't.
So what you have to do is something called "feature engineering," or "feature extraction," where you have to manually take all these things and create, essentially, a spreadsheet from them.
So basically, let's say that you have a bunch of birds and you're trying to build a bird classifier to figure out what kind of bird species it is.
You might actually have to take this picture, and then you have to measure the beak length, the wingspan, the primary color, and so on and so forth.
So you basically structuring the unstructured data manually.
And this process of structuring unstructured data is basically called-- we use the word "representation." We take the raw data, and we represent the data in a different form.
And the reason why I'm focusing on the use of the word "representation" is because it becomes really, really important a bit later on when we get to deep learning.
So we have to represent the data in a different way for it to work.

That's the basic idea.
All right, so what that means is that historically, researchers would manually develop these representations.
And once you develop them, once you have the representations, you can just use traditional linear regression or logistic regression and get the job done.
So the whole name of the game is the representations.
So in fact, people doing PhDs, for example, in computer vision, would spend, like, four years developing amazing representations for solving one particular little problem.
We have a bunch of, say, CAT scans, and we need to take the CAT scan and figure out whether a particular kind of stroke-- there is evidence for it in the CAT scan.
They might actually sit and develop all kinds of representations and test it and so on, and then they'll finally declare victory and say, yay, I'm done with my PhD.
Here is this amazing representation, and you can build a classifier with it to predict a particular kind of stroke with high accuracy.
So that's where the world was.
Now, as you can imagine, developing representations-- because it's so manual-- is this massive human bottleneck, and this sharply limited the reach and applicability of machine learning as you would expect.
To address this problem, a different approach came about, and that's deep learning.
So deep learning sits inside machine learning.
And deep learning can handle unstructured input data without upfront manual processing, meaning it will automatically learn the right representations from the raw input.
"Automatically" is the keyword.
Automatically learn representations, which means that you could give it structured data, you can give it pictures, you can give it text, you can give it anything you want, and just learn it.
It can automatically extract these representations.
And since it's being automatically extracted, you can imagine a pipeline where the raw data comes in, you have a bunch of stuff in the middle that's learning these representations automatically without your help, and then boom, you just attach a little linear regression or logistic regression at the end.
Problem solved.
That, in a nutshell, is deep learning-- input, a whole bunch of representations being learned, and then piped into a linear or logistic regression model.
So the amazing thing is this simple idea is just incredibly powerful.
That idea has led to ChatGPT, has led to AlphaGo, AlphaFold, and so on and so forth.
And I kid you not, I've been doing deep learning for about 10 years now, and every time I look at it, I literally get goosebumps every so often-- that something so simple could be so powerful.
It's really, like, boggles the mind.
I'm like, I'm just so lucky to be alive and working during this period.
And coming from people who have been in the industry a long time, this breathless exclamation is not very rare, particularly because I'm not in marketing.
I actually mean it, with all due apologies to various marketing folks.
[LAUGHTER] Just realized it's being taped, so.
So this demolished the human bottleneck for using machine learning with unstructured data.
And so it comes from the confluence of three forces-- new algorithmic ideas, a lot of data, and then, very importantly, the fact that we have access to parallel computing hardware in the form of these things called GPUs-- graphics processing units.
And these three forces came together, and they were applied to an old idea called "neural networks," and that's basically deep learning.
And I'll go through it very quickly, because obviously we're going to spend half the semester looking into this thing in detail.
So what's the immediate application of the ability to automatically handle unstructured data?

What is, like, the no-brainer application?
It's OK if it's obvious.
Tell me.
Sorry.
AUDIENCE: Image classification.
RAMA RAMAKRISHNAN: Right.
So image classification, yes.
So you can take an image-- a good example of unstructured data-- you can do some classification on it.
But more generally, what I'm getting at is that every sensor in the world can be given the ability to detect, recognize, and classify what it's sensing-- every sensor.
Because remember, what does a sensor do?
A sensor is just a receptacle for unstructured data.
A camera is a receptacle for unstructured video or unstructured still images.
Microphone-- unstructured audio.
So every sensor-- you can imagine taking a sensor and sticking a little deep learning system behind it.
And now, suddenly, what comes out of the sensor of the deep learning system, you can count, you can classify, you can detect, you can do all kinds of stuff.
In short, you can analyze, and you can predict.
And the way I'm describing it right now, you'll be like, yeah, duh, obviously.
But you know what?
This "obviously" thing is actually not at all obvious in terms of whether it will help you find interesting applications or not.
So here's something I literally saw last week.
Actually, I have another slide before that, but we are coming to that.
So for instance, every time you use Face ID, unlock your phone, this is the basic principle at work.
The camera and the iPhone is the sensor, and they stuck a deep learning system behind it to do image classification-- drama, non-drama.
That's what it's classifying.
And so here, you have a breast cancer-- this is a breast cancer detection system from a mammogram.
By the way, this picture is a very interesting picture.
So there's a professor in EECS, Regina Barzilay, who's a very well-known expert in this field.
And she actually has built a breast cancer detection system, which has been deployed at Mass General Hospital.
And it turns out, she's actually a breast cancer survivor, and she was-- --she's good now, all good-- but after she built her system, I heard that she actually ran that system against the mammograms from many years prior, when she went for a mammogram and was told that everything is fine.
She ran the system on that mammogram, and it came back and said, here's a problem.
So a very interesting example where a deep learning system picked up something that a radiologist could not.
So these things can be quite powerful.
Obviously, any self-driving system has numerous deep learning algorithms running under the hood-- pedestrian detection, stoplight detection, zebra crossing detection, so on and so forth.
It's being very heavily used in visual inspection manufacturing.
You have various cameras, and it'll show people looking at-- saying, OK, there is a dent or there's a scratch.
They have a little system, which is a dent detector, scratch detector, and so on.
That's going on right now.
And now I come to the example I saw last week, which is-- so this is an example of you can

create dramatically better products if you really internalize this idea of-- it's almost like you're looking at the world and saying, oh, there's a sensor-- can I attach a DL thing behind it?
That's the way you should be looking at the world for startup ideas.
So here's an example.
These apparently are the world's first smart binoculars.
This is the binocular two weeks ago, where you look at the bird, and now it tells you what kind of bird it is right there.
It's a simple idea, but imagine you are the first out of the gate with this feature.
You will have a little bit of an edge till everybody catches up, like, three months later.
Let's be very clear-- there are no long-term monopoly windows in the world.
There are only short-term windows, so the hunt is always on for a little monopoly window.
So here's an example of that.
So I encourage you to always think about the world as-- where are the sensors here?
And can I attach something behind the sensor to do something useful with it?
All right.
Now let's turn our attention to the output.
We have been talking about in structured data, unstructured data, and how deep learning has unlocked the ability to work with unstructured data, but you've been neglecting the output side of the equation.
So traditionally, we could predict single numbers or a few numbers pretty easily.
So you've all done the canonical-- should this person be given a loan application in machine learning?
So it just predicts the probability that a borrower will repay a loan based on a whole bunch of data or a supply chain.
You predict the demand for a product next week, or it can predict a bunch of numbers.
So given a picture, you can say, OK, which one of 10 kinds of furniture is it?
You can predict 10 numbers, 10 probabilities that add up to 1.
You can predict a whole bunch of numbers that don't have to add up to 1, such as the GPS coordinates of an Uber.
So these are all simple, unstructured-- sorry, simple structured output-- just a few numbers.
What we could not do very easily was to actually generate pictures like this.
We could not generate unstructured data.
We could only consume unstructured data.
You can generate text, you can generate pictures, and so on-- and audio, and so on and so forth.
So with generative AI, that problem is gone.
So generative AI is the ability to actually create unstructured data, and therefore, it sits within deep learning.
It still runs on deep learning, but it's just one kind of deep learning.
There's plenty of stuff going on in deep learning that's got nothing to do with generative AI.
Nowadays, of course, if you're a self-respecting entrepreneur who wants to ride this craze, you'll probably declare whatever you're doing as generative AI, and some VCs may actually be ready to fund you.
Who knows?
But the point is, there's plenty of stuff going on in deep learning that's got nothing to do with generative AI, but this is the overall picture.
Now, here we can produce unstructured outputs like pictures.
You can take this thing, and then you can actually come up with a nice picture-- description of it.
This actually is a very famous picture, by the way, in the world of computer vision.
So we are actually going to be analyzing this picture a little later on in the semester.
You can obviously go from a very complicated caption to an image.
You can go from text to music.

Can people hear it?
OK.
AUDIENCE: Yeah.
RAMA RAMAKRISHNAN: Yeah, yeah.
All right, and, of course, we go from text to text, i.e., ChatGPT.
And then as of a few months ago, things have gotten even more interesting where you can actually go-- you can send text and an image in, and you can get text out.
And in fact, as of a few weeks ago, you can send text, image, text, image, text, image in an arbitrary sequence into the system, and it can actually come back to you with text and image.
So things are becoming multimodal.
And I just want to share with you a really fun example I saw recently.
So this person sends this picture.
Can folks see this?
This is very complicated parking sign, apparently in San Francisco.
And they are like, it's Wednesday at 4:00 PM.
Can I park here?
[LAUGHTER] Tell me in one line-- because you really didn't want GPT-4 to be giving you a big essay about this.
You literally want to park.
So GPT-4 comes back and says, yes, you can park here for up to 1 hour, starting at 4:00 PM.
And folks, I double-checked this thing.
It's correct.
We all know these things hallucinate.
Can you imagine getting a parking ticket and telling the judge, I'm sorry, I didn't realize it was hallucinating?
So you have to double-check it.
So, yeah, so things are getting multimodal very quickly.
And so the picture here is that within GenAI, we used to have these separate circles-- text to text, text to image, text to music, text to this, text to that, and so on and so forth.
Those are all beginning to merge now inside GenAI because multimodal models are going to become the norm this year.
We already have really good closed models.
We actually already have very good open-source multimodal models, and so my feeling is that by the end of the year, the idea of using a text-only model is going to be like, really?
You do that still?
It's going to become like a quaint, old-fashioned thing.
I think multimodality is going to become the norm.
So that's where the world is, and this is the landscape.
So any questions on the landscape before we actually start doing some math?
OK.
Yeah?
AUDIENCE: It's not going to be the landscape, but it's more related to the breast cancer brush.
So how did she know that it was evaluating, like, the training that you do, and you were [INAUDIBLE], which is based on the existing breast cancer [INAUDIBLE].
But here, she was taking the mammograms from six weeks past, [INAUDIBLE], which is probably from a smaller version which was not there earlier, right?
And [INAUDIBLE]-- RAMA RAMAKRISHNAN: You mean the evidence of there being a problem would have been smaller?
AUDIENCE: Yeah.
RAMA RAMAKRISHNAN: Yeah.
AUDIENCE: So how do know your model is trained enough that it done [INAUDIBLE]?
RAMA RAMAKRISHNAN: Yeah.
So the question is that, in general, how do you train your model so that it gives you the right

answers, given that over the passage of time, the amount of evidence in this data could be very highly variable?
So in this particular case of the professor I talked about, everything at that point was going through an expert radiologist.
So five years ago, this mammogram was seen by a radiologist, and that person concluded there is no problem.
So that was the training label-- the wrong training label.
So typically what happens is that training labels could be wrong some small fraction of the time, so you need to have systems that are robust.
So your data needs to be complete, it needs to be comprehensive, it needs to be have correct labels.
If these ideas are not met, your systems are not going to be that good.
But as it turns out, with neural networks, even with some amount of noise in the labels, they still do a pretty good job.
So that's the general idea.
AUDIENCE: But there has to be some verification that [INAUDIBLE].
RAMA RAMAKRISHNAN: The verification comes from the human.
So remember, when we look at radiology data, the data we are working with is-- the input is, let's say, an image, like a mammogram or something.
And then a human radiologist, or a set of radiologists, have said, this has a problem or does not have a problem.
So that is called "ground truth." So it is this ground truth image and label-- this combination that's being used to train these models.
Yeah?
AUDIENCE: Will you be covering, like, embodiment [INAUDIBLE]?
RAMA RAMAKRISHNAN: Embodiment?
AUDIENCE: Yeah.
RAMA RAMAKRISHNAN: So are we going to cover embodiment?
So the embodiment here refers to the fact that if you have robots, they need to actually operate in the real world.
And so robots are an example of what's called "embodied intelligence." So unfortunately, due to the constraints of time, we're not going to get into robotics at all.
But I will say that a lot of the deep learning stuff we're going to talk about, those are all fundamental building blocks in modern robotic systems.
All right, so in summary, X and Y can be anything, and it can be multimodal.
I literally could not have put up this slide maybe two years ago.
So it's very simple in how it looks, but it's very profound.
You can learn a mapping from anything to anything at this point very easily, as long as you have enough data.
So now, note that all this excitement that we see around us is everything stems from deep learning.
Everything depends on deep learning, and so if you understand deep learning, a lot of interesting things become possible.
So let's get going, all right?
So we'll start with the very basics-- what's a neural network?
Now, recall logistic regression from back in the day.
So what is logistic regression?
You send in a bunch of numbers-- a vector of numbers-- and you usually get a probability out-- between 0 and 1, what is the probability of something or the other?
And so this logistic regression model is also represented in this form, if you will recall.
So basically, what we do is we take all these numbers, we run it through a linear function, you get a number, and then we take that thing and run it through 1 divided by 1 plus e raised to minus that.

And that's guaranteed to give you a number between 0 and 1, which can be interpreted as a probability, and that's logistic regression.
And the canonical loan approvals, things like that, all fall under this convenient bucket.
So this should be super familiar.
All right, now we're going to actually look at this simple, modest, humble little operation using the lens of a network of mathematical operations, and the reason why we do it will become clear a bit later.
So we'll take this very simple example where we have, let's say, two variables-- GPA and experience.
This is the GPA of some graduates, a number of years of work experience.
And then this is the dependent variable, which is either 0 or 1.
And 0 If they don't get called for an interview, 1 if they get called for an interview.
It's a two-input-variable, one-output-variable problem, and it's a classification problem because we are classifying people into-- will they get called for an interview?
Yes or no.
And so that's the setup for this problem.
And let's say that we actually run it through any-- and we actually try to fit a logistic regression model to it.
So if you're familiar with R, for example, you would use something like GLM to fit this model.
If you use something like statsmodels in Python, there's a similar function for it.
Scikit-learn-- there's another function for it.
You get the idea.
You can use whatever favorite methods you have for logistic regression modeling to get this job done.
And if you do that with this little dataset, you're going to get these coefficients.
The 0.4 is the intercept, 0.2 is a coefficient of a GPA, 0.5 for experience, and that is the resulting sigmoid function.
All right, cool.
So now let's actually rewrite this formula as a network in the following way.
So first, what we'll do is we'll take GP and experience and stick it here on the left side, and we'll put little circles next to them, and we'll call them the input nodes.
And so imagine that somebody writes a GPA into the circle-- 3.5, or years' experience, 2.0-- and then it flows through this arrow.
And as it flows through, it gets multiplied by its coefficient, 0.2.
The 0.2 is coming from here.
Similarly, experience gets multiplied by 0.5.
It comes in here.
And this node, as the plus indicates, is adding everything that's coming into it.
So it's adding 0.2 times GPA, 0.5 times experience, plus the intercept, which is the green arrow coming from-- on its own.
It comes through here.
And what comes out of-- this is just a single number.
And that number goes into this little circle, and then out pops a probability.
So I've sort of done this ridiculously long-winded way of writing a simple function, and the reason why I'm doing it is will become clear in a second.
So this is a little network of operations for the simple function.
And so, for instance, how you would use it is, to make a prediction, you will-- let's say someone has a 3.8 GPA and 1.2 years of experience.
You just plug it in here, do the math, you get 0.76.
Same thing here.
Comes in here, add them all up, you get 1.76.
You'd run 1.76 through the sigmoid, you get 0.85, and that is the probability that that particular individual may get called for an interview.

At this point, we're just doing logistic regression.
Nothing more complicated.
So now, if you have many variables, not two variables, like x1 through xk, the same logic applies.
Each one has some coefficient, and then there's an intercept.
They all get added up here, run through a sigmoid, and out pops this number.
Notice how the data flows from left to right.
All right, any questions on this?
All right, good.
So now a terminology.
So you will actually-- you'll discover that the world of neural networks and deep learning has its own terminology.
They have their own ways of referring to things that the rest of the world has been referring using something else for the longest time.
It's kind of annoying sometimes, but it's the way it is.
So remember, in regression, we used to call those numbers next to each variable as "coefficients" and the constant thing as an "intercept." Well, guess what?
In this world, those coefficients are actually called weights, and the intercept are called biases.
So in the neural network world, these are called weights and biases.
And sometimes if you're a little lazy, you may just call the whole thing as weights.
So when you see in the newspaper that, oh my god, this amazing model's weights have been leaked on the internet or on BitTorrent or something, that's what's going on.
All these coefficients have been leaked because once you know what the coefficients are and what the architecture is, you can just reconstruct the model.
All right, so that's what's going on here.
Now, why did we do this network business?
Why did we write it as a network?
Yeah, what is advantage?
Any guesses?
AUDIENCE: When you have multiple functions flowing through the network, it's just easier to see it that way.
RAMA RAMAKRISHNAN: Right.
If you have lots of things going on, it's easier to see it if you actually write it in graphical form.
Yes, correct.
But so is it only a usability advantage?
AUDIENCE: The other thing is you want different functions for different layers of the [INAUDIBLE].
RAMA RAMAKRISHNAN: Ah, OK, so maybe we want to use different functions in different layers.
But I think there's actually even a larger, more basic point, which is that the moment you write it down, you suddenly realize that I could have lots of things in the middle.
I don't have to go from the input to the output directly.
I could do lots of things in the middle.
That's some of the key idea.
So what you do is-- so remember the notion of learning representations of unstructured data, where you take a picture and say beak length and things like that.
And remember, I said deep learning actually automatically learns these things.
Where is that automatic learning coming from?
Well, this is where it's coming from.
So what we do is we take this thing-- this just a logistic regression model-- inputs get added up as a linear function or run through a sigmoid.
And then we are like, hmm, if we want to learn representations of the raw input, we better be

doing something in the middle here because the output is the output.
That's not going to change.
It's either a dog or a cat.
You don't have any choice as to what it is.
The only agency you have at this point is you can take the raw input and do things in the middle with it.
You can do a lot of stuff in the middle and then run it through something to get the output.
So in any mathematical discipline, if someone comes to you and says, here's a bunch of data-- I want you to do something with it-- what is the most basic first thing you should do?
Run it through a linear function.
The most basic thing in math is a linear function.
So given anything, just run it through linear function, see what happens.
So that's exactly what we can do.
So the simplest thing we can do here-- we can insert a bunch of linear functions.
So what we do is we take all this input and we just run it-- we do a linear function on it.
So think of this as x1 times 2, plus x3 times 4, and all the way to xk times 9, plus some intercept, and boom, it goes to the other end.
So this little circle here with a plus in it is just-- thank you-- this is just a linear-- it's a shorthand for a linear function.
So whenever you see a circle with a plus, it's just a shorthand for a linear function.
So you can take this whole thing and run through a linear function, and when you do it, you'll get some number right there.
You'll get some number.
So you've taken these k numbers and you've compressed them in some way into one number.
But you don't have to stop at one number.
You can do more.
So we can have a stack of linear functions in the middle.
There's a linear function here, another one here, another one here.
At this point, the k numbers you have-- k could be, for example, 1,000.
It's just the size of your input data.
You have taken these k things, and you have compressed them into three numbers at this point.
So maybe 4 is the right number, maybe 10 is the right number.
We don't know, and we'll get to how do we know what the right number is later on.
So we can stack as many linear functions we want.
So we have transformed this k thing into a three-dimensional vector.
k numbers become three numbers, and now we can flow these three numbers through some other little function.
And as you will see in a few minutes, that function is called an activation function, and it's chosen to be a nonlinear function because if you don't choose it to be a nonlinear function, all the effort we're doing is going to be a total waste of time.
For now, just take it on faith that you need to have nonlinear functions here, but note that the three numbers here are still three numbers.
They are three different numbers.
We did three numbers.
And once we do this, we'll be like, you know what?
This was fun.
Let's do it again.
So you can do it again, and you can keep on doing it.
You can keep it 100 times if you want.
And the key thing is that every time you do it, you're giving this network some ability, some capacity to learn something interesting from the data, to learn an interesting representation.
Now, of course, you're thinking, well, how do we know it's interesting?
How do we know it's a useful thing?

And we'll come to all that later on.
We're just giving it the capacity, the potential to learn interesting things from the data.
Whether it actually lives up to its potential, we don't know yet.
We'll give it the potential because the more transformations of the input data you make, the more opportunity you have to do interesting things with it.
If I don't even give you the opportunity to transform it once, you don't have any opportunity.
If I give you 10 chances to transform things, you have 10 shots at doing something useful.
So you can do this repeatedly.
And once we are done doing these transformations, we just pipe it through to our good old logistic regression sigmoid here, and we are done.
So this is the basic idea.
And so just to contrast it, this was good old logistic regression, where we take the input, we run it through a linear function and pop out a number-- a probability number.
But after we do all this stuff, the input stays the same, the output stays the same, but in the middle, we just run it through a whole bunch of these functions, these layers-- boop, boop, boop, boop-- and then we get the output.
That's all we have done, and this is a neural network.
A neural network is nothing more than repeatedly transformed inputs which are finally fed to a linear or logistic regression model.
Any questions?
Could you use the thing so that everyone can hear?
Yeah.
AUDIENCE: There are two questions.
Firstly, so can we say that there is a challenge of explainability?
Is it that we don't know which arrow it went through?
That's one.
Second, who's controlling the number of iterations or the number of functions?
That's up to us?
RAMA RAMAKRISHNAN: Yeah.
AUDIENCE: Or, how does that work?
RAMA RAMAKRISHNAN: So, yeah-- so the first question-- explainability-- we actually know exactly for any given input data point-- we know exactly how it flows through the network, so there is no problem there.
The problem is in ascribing, OK, we think this person is going to be-- repay the loan because of this particular attribute.
We don't know that because those attributes all get enmeshed together and goes to this complicated thing.
So we know exactly what happens-- we just can't give credit to any one thing very easily.
And again, I'm just standing on the brink of this vast ocean of something called explainability and interpretability, which I'll get to a bit later on in the semester, but that's the quick kind of right-ish, kind of wrong answer.
Number two, we decide the number of layers.
We decide a whole bunch of things.
And as we'll see in a few minutes, there is something that's given to us and something we get to design, and I'll make it very clear which is which.
Yeah?
AUDIENCE: Yeah, so this-- RAMA RAMAKRISHNAN: Did I say your name right?
AUDIENCE: Yeah.
So which functions have to be linear?
And also, why does it have to be linear?
RAMA RAMAKRISHNAN: Yeah, so these functions-- the f of x here-- they have to be nonlinear.
As to why they have to be nonlinear, we'll get to that in a few minutes.
So these are called "neurons." These things where you-- basically, there's a linear function

followed by a little nonlinear function.
Each one of these things is called a "neuron." By the way, this is loosely inspired by the way how neurons work in human-- in mammalian brains.
But the connections between neuroscience and deep learning are very heavily argued, so I'm going to stay away from it.
Suffice it to say, I just think for building practical deep learning systems in industry, you're not worry about this.
All right, let's move on.
Terminology-- this vertical stack of linear functions or neurons-- this vertical stack is called a "layer." This a layer, that's a layer.
And these little nonlinear functions, which we haven't got into yet, are called "activation functions," and we'll get to why they are called that in just a second.
And the input is called an "input layer," and I have the word "layer" in double quotes because it's not really doing anything.
It's just the input, but we call it the input layer.
And the very final thing that produce a output is called the "output layer," obviously, and everything in the middle is called a "hidden layer." So the final piece of terminology is that when you have a layer like this in which, say, three numbers are coming out and there's another layer, if every neuron in this layer is connected to every neuron in this layer, it's called a "fully connected" or "dense layer." So for instance, here, this arrow that's-- whatever number is coming out-- let's say the number 3 is coming out of this thing here-- that number 3 flows on this arrow to this thing, flows on this arrow to this neuron, and flows on this third arrow to this neuron.
That's what I mean.
So every neuron-- its output is being sent to every neuron in the following layer.
We call it fully connected or dense.
And then if you look at logistic regression-- this just logistic regression-- you can see, basically, logistic regression is a neural network with no hidden layers.
So in some sense, logistic regression is almost the simplest possible network you can think of-- like, barely a neural network.
It's got no hidden layers.
That's what makes it logistic regression.
And so as you might have guessed by now, deep learning is just neural networks with lots and lots of-- of what?
AUDIENCE: Layers.
RAMA RAMAKRISHNAN: Yes, layers.
So here are a few.
And by the way, these are not even considered all that impressive these days, but I put them up because this thing here is called "ResNet," and it's famous because the ResNet neural network was, I think, the first network to surpass human-level performance in image classification.
It's sort of like the Skynet [CHUCKLES] of image classification.
It surpassed human-level performance.
And I'm putting it up here because we will actually work with ResNet on next Wednesday.
And we'll actually take ResNet, we'll fine-tune it, and solve a real problem in class.
All right, so it's got lots and lots of layers.
Now let's turn to these activation functions.
We've been ignoring these little guys so far.
So the activation function at a node is, first of all, it's a function that receives a single number and outputs a single number.
It's not very complicated.
It receives, basically, this-- this is a linear function, which receives all these inputs-- it could be 10 inputs, 1,000 inputs-- runs it through a linear function, outputs a number.
And that single number-- a scalar-- goes in here, and it comes out as another single number.

Just remember that.
And so these are some of the most common activation functions.
In fact, the sigmoid we saw, which is actually be used for the output, is actually a kind of activation function where a single number comes in and it gets mapped into this curve because of this thing.
So the single number that comes in is a, and it gets transformed as 1 divided by 1 plus e raised to minus a.
And you get a shape like this, and it's called the "sigmoid activation function." And as you can see here, for very small values, for very negative values, it's going to be pretty close to 0, meaning it won't get activated.
And for very, very large values, it's going to be pretty close to 1.
All the action happens in the middle.
When your values are somewhere in this range, there's a dramatic increases in what comes out, so that little thing in the middle is the sweet spot for these functions.
And this-- I'm almost embarrassed to call it an activation function because it's literally not doing anything.
It's sort of gettinig a nice label for free.
Basically, it says you just get a number, you just pass it straight along.
It's a linear activation function, but just for completeness, I want to put it here.
And then we come to the hero of deep learning, which is the rectified linear unit.
Rectified linear unit-- it's called ReLU.
And ReLU is going to become part of your vocabulary very, very quickly.
And so ReLU is actually a very interesting function.
So you write it as maximum of whatever number and 0, which is another way of saying if the number is positive, just send it along unchanged.
If the number is negative, send a 0 instead-- squash it to 0.
So which means if the number is negative, nothing happens.
If the number is positive, it breaks up.
So what happens is that you could have a very complicated linear function with millions of variables, and then it puts a single number, and that number, unfortunately, happens to be negative-- the ReLU is not impressed.
It's going to send a 0 out.
It's a very simple function.
And many, many folks who have been in deep learning for a long, long time believe that the use of the ReLUs is one of the key factors that led to the amazing success of deep learning because it's got some very interesting properties, which we'll get to, hopefully, on Wednesday.
So the shorthand here is that whenever you see this thing, it's just a linear activation-- linear function followed by just sending it straight out.
If I put a ReLU in here, I'm going to denote it like that, which mimics the graph-- how it looks.
And if I put a sigmoid, I'm just going to use this thing here.
Just a visual shorthand.
There are many other activation functions, by the way.
There's something called the tanh function, the leaky ReLU, the GELU, the swish.
I mean, it's like a menagerie of activation functions because very often researchers will be like, well, I don't like this activation function.
Here's a little modified version of the function, which is going to be better for certain things.
So people's research creativity is, at this point, has gone unhinged, so lots of options.
But if you just stick to the ReLU for your hidden layers, you can basically get anything done, practically.
You don't have to worry about anything else.
So we'll only focus on ReLUs for all the intermediate stuff.

Yeah?
AUDIENCE: Yeah, how do you gauge which activation function is more suited for your use case?
RAMA RAMAKRISHNAN: Yeah, so the rule of thumb here is that for your hidden layers, use ReLUs because empirically, we have seen that they do an amazing job.
For your output layer-- your very final thing-- you actually don't have a choice because what you have to use depends on what kind of output you have to work with.
If it's an output which is a probability number between 0 and 1, you have to use a sigmoid.
If it is, say, 10 numbers, all of which have to be probabilities and they have to add up to 1, you got to use something called the "softmax," which we'll get to on Wednesday.
So it really depends on the output, and the nature of the output dictates what you use in the output layer.
So coming back to this-- so if you want to design a deep neural network, the input is the input, the output is the output, and so you get to choose everything else.
You get to choose the number of hidden layers, the number of neurons in each layer, the activation functions you're going to use, and-- for the hidden layers-- and then you have to make sure that what you choose for the output layer matches the kind of output you want to generate.
So this is all in your hands.
You decide what happens, but you will-- there's a lot of guidance for how to do these things, which we'll cover as we go along.
Did you have a question?
AUDIENCE: Kind of, but I guess I'll do it.
Is there also exploration in dynamic setting of layers so that our user is determining the layers that the system itself [INAUDIBLE] layers.
RAMA RAMAKRISHNAN: Yeah, so there is a whole field called neural architecture search-- NAS-- where we can actually try a whole bunch of different architectures and then use some optimization-- and, in fact, reinforcement learning, which we won't get to in this class-- as a way to figure out really good architectures for any particular problem.
But the question of, OK, when I'm training a model with a particular kind of data, the first pass through the training data, I'm going to use two layers-- the second pass, I'm going to do 7 layers-- that is not done.
And the reason it's not done is because of certain other constraints we have and how we can do the optimization and the gradient descent and stuff like that.
But what you can do-- and we will look at this thing called "dropout"-- for certain layers, you can actually, for each time you run it through the network, you can decide on this layer, I'm not going to use all the nodes.
I'm going to drop out a few other nodes randomly.
And it's a very effective technique to prevent overfitting, and we'll come to that a little later on.
Yeah?
AUDIENCE: One question regarding neural networks is about the coefficients.
If it's a small size, is it fine on its own?
Or do we have as the coefficients [INAUDIBLE].
RAMA RAMAKRISHNAN: No, the whole trick here-- the whole name of the game-- is we use the training data and something called a "loss function," which I'll get to on Wednesday, along with an optimization algorithm so that the network figures out by itself what the weights need to be, what the coefficients need to be so as to minimize prediction error, and that's the whole thing.
The magic here is that we don't have to do anything.
We only have to set it up, sit back-- often for many hours-- and watch it do its thing.
Yep.
AUDIENCE: Just one last question.
You mentioned nodes just now when you were answering Rolando's question.
Can you just confirm exactly what a node is?

I have an idea that it's basically any circle, but you just added a lot more detail.
RAMA RAMAKRISHNAN: Sure.
No, when I'm referring to a node, I'm literally referring to something like this, which think of it as a linear function followed by a nonlinear activation.
So it receives a bunch of inputs, runs it through a linear function, and pass it through, like, a ReLU or a sigmoid or something, and out pops a number.
So in general, a node will have many numbers potentially coming in but only one number going out.
Now, that one number may get copied to every node in the next layer, but what comes out of that particular node is just a single number.
All right, so let's use a DNN for our interview example.
So in this problem, we had two inputs-- GP and experience.
The output variable has to be between 0 and 1 because you have to predict the probability that someone will get called for an interview.
So the output size is fixed-- sorry-- the input size is fixed, the output is fixed.
So since it's really only the very first network we're actually playing with, let's just start simple.
We'll just have one hidden layer, and we'll have three neurons.
And as I mentioned to Tomaso's question from before, if you are choosing activation functions in the hidden layers, just go with the ReLU as a default.
It usually works really well out of the box, so we'll just use a ReLU.
And since the output has to be between 0 and 1, we don't have a choice.
We have to use a sigmoid for the output layer.
That's it.
So those are our design choices.
And when we do that, this how it's looked like.
We have two inputs-- x1 and x2, GP and experience-- and then it goes through these three ReLUs, and then out comes these three numbers, and they pass through a sigmoid, and we get a probability y at the end.
All right, quick question-- concept check-- how many weights-- how many parameters, both weights and biases, does the network have?
Let's take a moment to count.
All right, any guesses?
Yeah?
AUDIENCE: 12.
RAMA RAMAKRISHNAN: I think you're almost there.
Are folks going to be doing a non-binary search on this now?
OK.
AUDIENCE: Is it 9?
RAMA RAMAKRISHNAN: No.
Yes?
AUDIENCE: 13.
RAMA RAMAKRISHNAN: Yes, very good.
So that's 13.
And my guess is that the reason you came up with 12-- and I made the same mistake-- that's why I know it-- is so you probably forgot this green thing here.
So what folks often forget is the bias.
We all count the things.
And the easy way to do it is say, OK, two things here, three things here, 2 times 3 is 6, 3 times 1 is 3, another 9.
And then you have to add up all the intercepts, so you get 13.
And so when we get to very complicated networks-- the first two or three times you work with very complex networks-- and we'll do it starting very soon-- just get into the habit of

hand-calculating the number of parameters just to make sure you understand what's going on.
Once you get it right a couple of times, you don't have to do it anymore.
The first couple of times, hand-calculate to make sure you get it.
So, yeah.
So let's say that we have trained this network using techniques which we'll cover on Wednesday, and it comes back to you after training and says, OK, these are the optimal-- the best values for the weights and the biases that I have found.
So now your network is ready for action.
It's ready to be used.
And so what you can do is-- let's say that you want to predict with this network.
If you have x1 and x2, what comes out of-- so what comes out of this top neuron?
Let's call it a1.
It's basically this.
That's what's coming out of this thing.
For any x1 and x2, this is what's coming out, similarly for a2 and a3.
And then what comes out at the very end is basically a1 times that, plus a2 times that, plus a3 times that, plus 0.05.
And the whole thing gets run through the sigmoid, and that's what you get.
So this slide and the one before, just make sure you look at it afterwards and-- to make sure you totally understand the mechanics of it because this is really important.
And if you don't fully understand, like, internalize the mechanics, when we get to things like transformers, it's going to get hard.
So just make sure it's, like, automatic at this point.
It should be reflexive.
So, yeah, and so when you want to predict anything, just run some numbers through it.
You get all these things, and boom, you calculate it.
It turns out to be 22.6.
That's the answer.
All right, so I just want to say that-- let's say that you built this network and now you're like, hey, given any x1 and x2, I can come up with a y, but I'm feeling a little mathy.
Can you actually write down the function?
Yeah, you can write down the function.
This is what it looks like.
Super interpretable, right?
So this goes to the comment that, you made earlier on, where the act of depicting something using this graphical layout makes it so much easier to reason with and to think about compared to trying to figure out what this function is doing.
The other point I want to make is that this contrast-- what we just saw with the logistic regression thing we saw earlier, which was this little function-- and so here, even this simple network with just three hidden layers-- sorry, three nodes in that single hidden layer-- it's so much more complicated than the logistic regression model-- so much more complicated.
And it is from this complexity springs the ability of these networks to do, basically, magical things.
And that's where the complexity comes from.
That's where the magic comes from.
And here, in this case, the number of variables hasn't even changed.
It's still only two.
But we can go from the two inputs to the one output in very complicated ways as long as we know how to train these networks the right way.
That's the secret sauce, which we'll spend a lot of time on.
So, yeah, to summarize, this is what we have-- it's a deep neural network.
By the way, this kind of network where things just flow from left to right is called a "feedforward neural network," in contrast to some other kinds of networks called "recurrent

networks," which we won't get to in this class because transformers have actually proven to be much more capable than recurrent networks.
And those have become the norm, so we'll just focus on those instead.
And so this arrangement of neurons into layers and activation functions and all that stuff-- this called the "architecture" of the neural network.
And as you will see later on, the transformer-- the famous transformer network-- is just an example of a particular neural network architecture, much like convolutional neural networks, which we'll get to next week for computer vision, or another example of a particular network of the architecture.
So we will focus on transformers.
They are a particular kind of architecture.
All right, so in summary, that's what we have.
You get to choose the hidden layers and neuron's activation functions and stuff like that.
The inputs and the outputs are what you have to work with.
And so we will actually take this idea and then use it to actually solve a problem from start to finish on Wednesday.
So I think I'm done.
I give you three minutes back of your day.
Thank you.
[APPLAUSE]