[SQUEAKING]

[RUSTLING]

[CLICKING]

SARA BEERY: Let's get started. So today, we're going to kick off the first of a little two lecture mini series on transfer learning, specifically how we think about taking information that might have been learned from a model and then transferring it to some new task, some new challenge, maybe some new distribution. We talked about distribution shifts last week.

So at a high level, the idea of transfer learning is that you want to try to learn even if you don't have very much data. And so we've tried to structure this framing of transfer learning into these different sub areas. So first, we're going to talk about how you can maybe transfer knowledge from a machine learning model that you have about the mapping from inputs to outputs.

And so within that, we're going to talk about fine-tuning, and then we'll also talk a little bit about domain adaptation. And then we'll talk about transferring knowledge about the outputs, and so in that case, we're going to be discussing knowledge distillation.

And then we're going to be talking a little bit about foundation models and maybe what this new paradigm is in terms of transfer learning, kind of introducing this idea of in-context learning or prompting as a mechanism for transfer learning by just asking. And then next Tuesday, we're going to finish this up with one further discussion of how we might transfer knowledge about the inputs of the data.

So why deep learning? This is a slide from Andrew Ng, who, of course, I think has a very, very famous lecture series on deep learning. And here, he essentially is pointing out that deep learning might be different from older or standard learning algorithms specifically in terms of how it scales with the amount of data that you have to train on. It might be a little bit hard to see with the lights as they are.

But essentially, the idea is that many learning algorithms, older learning algorithms, saturate. With some amount of data, you just don't see performance continuing to improve. But we have really not seen that type of saturatior as we scale the amount of data that we're giving these deep learning algorithms.

And next Thursday, you guys will have a lecture from Phil around some of these scaling laws, these relationships that people are starting to learn about the scale of the number of parameters, the amount of available data. But one kind of high level analysis that was done seemed to suggest that with the current capacity of machine learning models we have, we basically have far more capacity than data.

So essentially, the idea that there's a lot more room to scale on the data dimension still before we're really hitting the capacity of our largest deep learning models. And that's talking about billion data point data sets, things that are on the order of scale of the entire internet. We're still not hitting the actual computational capacity of these models.

So few-shot learning is the idea that you can maybe learn even if you have very little data. So it turns out that humans are shockingly good at this. We're really good at recognizing patterns and translating them even to things that we've never seen. So here, if I tell you that symbol up at the top is a dax-- these are made up. This isn't real. But if I tell you that's a dax, can you guys tell me which of the ones below is also a dax? Yeah.

AUDIENCE: Second row, second from the right?

SARA BEERY: Second row, second from the right. Yeah. Right? It looks like a dax. OK. What about this? Which on the bottom is an example of the same concept as the item in the box? Someone else. Yeah.

AUDIENCE: Second row, first to the right.

SARA BEERY: Yes. So those are both Segways, right? But then this kind of begs this interesting question, because there's potentially ambiguity here. If you only have this single example, and you're telling me which is the same, what you made was the guess that was probably most similar because the orientation of the wheels and the structure. But maybe the thing I was looking for was instead, actually, I wanted you to tell me all the things that had two wheels instead of four or one.

And so one of the things that's challenging when you're doing this few-shot learning type of scenario is with less data, it can be harder to understand without ambiguity what the boundaries of maybe the categories of interest are. So if I'd showed you three examples, one was a Segway, one was a scooter, and one was a bicycle, and I said, OK, find me the ones that are also in this category, it would have been much easier for you to understand, OK, she's looking for things with two wheels.

Or at some different level of granularity, maybe I was trying to just learn wheeled things, in which case, I might need to have, again, more diverse examples that cover the space of the concept I want to learn. So few-shot learning really is just this idea that we can learn to maybe generalize from our past lived experience how to recognize similarity and patterns in a useful way.

So up until this point, a lot of what we've talked about with deep learning is, essentially, you're going to have a data set, and you start with an empty slate. And then you're essentially trying to learn some physical rules, some concepts, some dimensions of similarity that are useful for a specific task.

But what humans do instead, when they try to learn something new, is we really have a lot of prior knowledge coming from our experience in the world. From the time that we're incredibly young, we're physically embodied in the world, learning about concepts like physics, like structure. We're learning through the entirety of our education and just in our experience, in our lives, how to think about these measures of similarity and difference.

And different people can have different mental models of the world. This actually comes up a lot currently when you're thinking about scaling large scale data labeling, that, actually, if you're getting data labeled from people from different cultures, they might actually have different mental models than the one that you intend in your culture. So there's this idea of, my mental model is based on some prior, and we don't necessarily know-- or there really isn't a concept of which mental model of the world is correct.

But these mental models are built, and they kind of capture representations of things, kind of models we have of how the world operates, maybe different skills. And that means that we're basing anything we learn on top of a bunch of prior knowledge, basically like systems that are really well posed to learn new things. And actually, a lot of what you're doing in college is learning how to learn. You're learning how to learn new things efficiently. And so there's this question of, how can we give deep learning models or deep networks similar access to prior knowledge?

And so a very, very simple way to do that is just pretrain them on other tasks or on prior tasks. So the idea that you can send your neural network to school, and you can try to get it to learn how to learn, and the way that you might do that is maybe training on a huge amount of data with some other set of categories that might not be the thing you're really explicitly interested in, but it's things that you have data for at scale.

And then you can use some mechanism to transfer that knowledge from the pretained net so that you can solve new tasks. And so one view of this idea of learning from little data or transfer learning is actually the point of deep learning is to enable us to be able to do this type of problem solving with little data. So you're trying to learn these representations, these models that have a good sense of what's useful. What is a good thing to build on top of if we want to learn?

So that's kind of the motivation for talking about transfer learning, right? So when we think about a model in general, usually, we have some big system, and it will map some set of inputs to some set of outputs in some very fun, complex, nonlinear way. But we're thinking about, what actually are we transferring? And so you can think about transferring some knowledge about the relationship, for example, from the inputs to the outputs.

You could also think about transferring knowledge in different ways. We can think about transferring knowledge about biases in a model. You can think about transferring knowledge about an optimizer, perhaps. How do we optimize models? We can think about maybe transferring knowledge about potential error modes or ethical risks or societal implications of models.

So we'll really be focusing on pretty simple mechanisms to think about knowledge transfer, but I do think that there's a lot of more broad sets of transfer or this idea of sharing information in a valuable way that are really not understood or not solved.

So one thing that we still haven't really figured out how to do is building an agent that's able to actually learn some intrinsic motivation about how to explore in the world, which then could be transferred to a new agent, which would let them explore more efficiently. This idea of transferring motivation within a robotic agent.

So there's a lot of ways that we have these kind of brute force knowledge transfer mechanisms that we'll talk about. But I think there's a lot of space still to explore in the machine learning research arena, just in terms of the best ways to try to share information and transfer knowledge. Yeah?

AUDIENCE: How much of that is limited-- my very basic understanding of some of the challenges in, say, reinforcement learning is that it's more of an issue of sample efficiency. And thinking back to 2012, one of the breakthroughs wasn't really that the algorithms changed, but just that they finally got it [INAUDIBLE] GPU, and it sped up the training process. How much do people think that the bottleneck is the algorithm or the ideas around how to do this versus just we're not computationally efficient enough?

SARA BEERY: So the question is basically, what's the bottleneck in terms of knowledge transfer? Is it fundamentally algorithmic, or might it be related to, instead, computational efficiency? And you brought up a really great example of essentially how figuring out how to use GPUs effectively was a huge breakthrough in terms of progress across many different areas of deep learning, reinforcement learning being one of them, but basically anything else as well.

I think it's a little bit of everything. And this question of sample efficiency, I would not say that we have good algorithms for sampling data efficiently or for learning from that sample data efficiently. I mean, we have algorithms that do it, but I feel like there's a lot of room to improve.

Cool. So we're first going to talk a little bit about this idea of transferring knowledge from the mapping, so essentially, transferring knowledge from the model you've learned itself. How you go from inputs to useful outputs? So a simple example of this would be something like, OK. I have a lot of data on music with associated genres, and the idea being that being able to recognize a genre is potentially useful.

It's teaching models something about the structure of music, something about what makes different songs similar. Two things being of the same genre argues that they might be similar in some way. And then maybe what you want to then adapt or transfer that knowledge to is something where you have a lot less data that specifically predicting, let's say, your individual preferences for songs.

So maybe you can do genre recognition on the entire cadre of music on Spotify. And then if you actually want to learn how to predict what songs you're likely to enjoy, that you're going to prefer, you might only have a much smaller set of data, which is all the songs you've ever listened to, which is, of course, not even close to all the songs that exist.

And then the actual testing would then be taking of fine-tuned model-- you can see they have this f prime-- and using it to actually predict on new music whether you're going to it or not. And then-- I don't know-- see, do you like the music that it's suggesting for you?

And so this is kind of that broad, simple thing in terms of what we call fine-tuning. And at a very high level, it's that you're pretaining your network, your set of weights and biases, on some task that's a pretaining task. And that will result in some parameters for your model, W and b, weights and biases. Then you initialize a second network with some or all of W and b, and so there's these kind of interesting question of how much knowledge about the mapping you want to transfer.

So usually, maybe the very final layer of a model, that's going to be pretty specific to the pretaining task. So often, it doesn't really make sense, especially if the output shape is not the same. You cut that off, you add some different final layer on, but maybe, you really only want those early layer features to be captured. And so you actually let the model-- you initialize it with less of the initial model.

And then you think about training that second network on some task b. And so now, you'll get some slightly adjusted parameters, W prime and b prime. And the idea is then you have this learned representation that is essentially an encoder, and that's what we're transferring. So you're taking the representation learned from the initial model that's encoding something useful about your data, and then you're transferring that and then learning how to make use of that useful representation on something downstream.

And we'll talk a little bit about some of these different empirical choices you can make here around how much you let the model adapt, how much you let it change from the initial representation. But empirically, actually, as long as the types of input data are similar-- so images and images, for example-- what we often find is that a lot of the structure in the models, like weights and biases, actually stays the same, particularly in those early layers after you've done the fine-tuning.

So even if you haven't explicitly constrained the model not to change that very much, what you can do is keep a learning rate low or freeze some of the network. But if you don't do that, you actually find that a lot of the structure will stay the same, which does seem to argue that some of that is just universal in some sense. It's universally useful for, let's say, images or audio.

So we previously saw a very specific case when we were talking about similarity-based representation learning where we talked about self-supervised pretaining. So here, with this self-supervised pretaining, we're not necessarily explicitly or directly training the model to do something we might care about. Instead, we're just training the model to learn something about data similarity.

And then that ends up-- it has been shown-- to be quite useful for training representations that are good for downstream tasks we might care about. And one of the main benefits of this type of self-supervised pretaining is the fact that, because it doesn't require labels for data, it's making that pretaining on a lot of data-- like a lot, a lot, a lot of data.

The amount of data that exists without any specific label is much larger than any of labeled or clean data set. So here, you think about contrastive pretaining and then maybe you're going to test it on some of new recognition task. But instead, you could think about fine-tuning a model, which is how you take those initial representations and transfer them to then actually learn how to do that new recognition task.

OK. So now, I'm going to get into a couple pretty simple but pretty practical considerations. So what if you're pretaining on some images, and you're predicting whether they're wheat or corn? And then now, we want to fine tune our model. But now, we have an x prime and a y prime that are not the same.

So essentially, we have a model, and our initial model was trained on images that are RGB. And it was trained to predict two categories as the output. And now, the model we actually want is going to actually take in images that are just grayscale. So we only have a single dimension, and we want to predict on outputs that have an additional category. So the structure of the model itself is actually a bit different.

Essentially, if you're changing either your input dimension or your output dimension or both. So one kind of very simple mechanism that you can do is you can add kind of like a glue into the model, which is just a way to transfer data through to get it into the right format.

So then when you're doing your pretaining, if you know that your downstream task is not going to have necessarily the same structure, you define these new z's, which are maybe models, or even just like a few simple layers, fully connected layers, what have you, that enable us to then map each input and output to the same dimensionality.

And so you would do this where you need it, right? If you still are going to have RGB images, but then all you're doing is adding one more output dimension, well, then one thing you can do is you can take the final output layer of your initial model, and you can learn some reprojection to project it into more dimensions.

Or you can do what I talked about sooner, and you can consider this z to be like that final mapping from your representation, that encoder, to the number of categories you want to predict. And then you can just swap that out and maybe learn some different one that's now in this new model. So you're just essentially defining points within the structure of your model that give you a bit of flexibility.

So now, this f1 goes to f1 prime. Those will be different from each other. f3 goes to f3 prime. Those will be different from each other in terms of their structure, like the actual number of layers, weights, or size. But then the f2 can stay the same. So the vast majority of the structure, you're not actually changing.

And this actually comes up quite a lot, particularly for some types of data that have maybe more input dimensions. So most image models are pretained on things like ImageNet, or these days, LAION, where you have huge amounts of RGB data. But then say you're someone who's working in robotics, and you have RGB-D. You have this additional depth dimension.

It's often true that you can still get a lot of value in doing transfer learning from models trained without depth, but now, you have to figure out how to add this additional depth dimension during your fine-tuning effectively. And so often, the way people will do that is they just learn some of additional encoding that takes that and maps it into three dimensions.

Or sometimes, depending on how much data you have, whether you can really learn that type of mapping, you do some of strange, just very simple model of how to transform one input dimension into another.

So if you're working with very high scale hyperspectral satellite data, and you have models that are pretained with three channel inputs, sometimes, people will just run something like PCA and take the principal components, the three most informative principal components, over all those hyperspectral bands, and use that as their three channel input when they're fine-tuning because the ability to actually learn all of those 380 different input layers effectively is, one, computationally very expensive, and two, if you don't have a lot of data, that's a lot of parameters to learn.

And so that can be tricky. Yeah.

AUDIENCE: So we're talking about adding or changing the stacks and the [INAUDIBLE]. Do you ever add stuff into this dimension for fine-tuning, or is that just--

SARA BEERY: In the width-- so the input size would change?

AUDIENCE: Basically, say if you had f2, adding additional channels to be parallel to f2.

SARA BEERY: Like concatenating?

AUDIENCE: Kind of.

SARA BEERY: Yeah. I mean, so definitely, a lot of these models will assume some of fixed input size. And then, say, maybe you want to train the model on a bigger input size. That's another way that you can think about building this little f1 that will take a larger input size image and then learn how to get it to that smaller input size that the pretained model was trained on. But I think the only way to think about changing the width dimension is in terms of the spatial extent, right?

AUDIENCE: So just in terms of input and output, not in terms of the throughput of the pretained models.

SARA BEERY: Oh, you mean like actually changing the structure of the model itself?

AUDIENCE: I guess like stacking on top of [INAUDIBLE].

SARA BEERY: Yeah. I mean, so one of the things we're trying to get across here is that you can treat these things as modular building blocks, right? And so you can take a building block that someone else has built and then stick it in.

And then you have to figure out what blocks you need to add to the system to maybe get that building block to be useful for you. But the point being that a really well-trained building block-- if that analogy still is making sense. A really well-trained building block is still useful, and it's maybe worth coming up with this modular structure that will make that work for your problem.

But as soon as you start actually trying to mess with that building block, like changing the weights and the biases within that block, it gets more complicated. There are people who look at doing it, and actually, the entire area of model compression is kind of changing that structure by removing stuff to make it more efficient.

And we'll talk a little bit about that later. In terms of adding more stuff, I don't know. I haven't seen a lot of work in that space, but I think as soon as you start adding more stuff, the value of the pretaining starts to go away, if that makes sense, because then you've changed the structure of the mathematical formulas.

Cool. So essentially, we're just trying to figure out how to make things useful. And so in practice, we'll essentially just learn mappings for any new stuff we add to make this work for our data. And so those new things would need to be then initialized from scratch, but that's a lot less parameters that need to be learned from scratch than if you're going end to end from scratch, starting from that blank slate.

And if you really don't have much data, you can also think about copying weights for that initialization. So for example, in that RGB-D case, sometimes what people will do is they will take the RGB channels that were learned from the original model, and then they'll staple on an additional channel, and they'll initialize it with, for example, just the blue channel parameters, with the idea that maybe there's some value in what's been learned for another channel that might still be transferable.

Again, this is all kind of-- no one really knows. No has a perfect set of heuristics for, if this is your data, this is the best thing to do. And this is really where the idea of designing a benchmark for your data set where you understand how to do comparison across some of these different choices so you can try things and see how they're affecting performance of your model-- that's where it starts to become really important.

Because most of the time, if it's some custom problem, some little dimension of the system will probably be somewhat customizable. And it's hard to say what's the best structure for a system without actually doing some testing.

So one of the issues is that if you only have a little bit of data, and you let the entire model train, and you train it until that loss is all the way saturated-- we don't have anything else we can learn-- we run into that catastrophic forgetting problem again, that thing we talked about a bit when we were talking about sequences, where it's like, somehow, the thing we learned from the original data, we learned too long ago.

The model's weights get changed too much, and you essentially lose capacity. So this definitely happens more with smaller models than big models, interestingly. But in order to avoid kind of overfitting on a really small amount of data for your specific task, there are a few different mechanisms that people use to try to strike that right balance, learning enough about the new task that you can specialize while not forgetting too much about this general useful representation.

And so a couple of things. You can think about freezing most of the network. So basically saying, I just want to keep most of this model exactly the same, and only training, for example, the final layers or so, those initial things at the beginning that you need to have that are new. You can explicitly use really small learning rates. So basically, just not letting the model move too much in any direction at any training step.

You can use what's called early stopping, which is-- essentially, you use a validation set, and you make a decision about which model weights to keep based on the best performance on your validation set with the idea that, essentially, you just haven't let the model train too much, so you stopped it before it can forget too much about the original pretext task.

Or another mechanism-- and one that tends to work pretty well, though, of course, there's some complications in terms of data balance that you have to strike, so sampling effectively-- is to just continue to train on the original data as well. So essentially, if you have access to the original training data, then as you're fine-tuning, continuing to add some of that pretext or some of that original training data is good.

But of course, there's a complexity here, which is, all right, but if you have different input types and different goals, this becomes harder and harder to do. If you were doing contrastive pretaining, and now, you're trying to do classification on top of those representations, still continuing to have the contrastive pretaining is a little bit more complicated.

But a lot of these choices, like these types of things, assume that you have some sort of good representative validation set, that your benchmark is well posed for your task. This might not always be possible, where you might see distribution shift going forward. And so it really does become quite important to think about measuring relative performance as close as possible to your representative problem. If you want to actually use the model on some data, defining measuring progress is just difficult and complicated and important.

And this is-- now, I'm going to be on a soapbox a little bit. But one of the things that drives me absolutely crazy about modern machine learning is that we're obsessed with the idea of bolded numbers and tables. And so this kind of begs the question that you can represent everything about what a model does with one single number that captures every dimension of importance to that model.

And I don't think that's true. I think we need nuance. And honestly, usually, people will-- this is bad behavior. People will pick the metric that their model does best on. They're like, oh, this makes us look the best, so we're just going to show that one in our paper, and we're not going to talk about the fact that for some of these other metrics, we don't do as well.

I really love it in a machine learning paper when people are honest about their flaws. Where you're like, hey, all right. We see some really great progress in these sections. And then you have a really nice limitations section that's realistic and captures the dimensions of where there still is space to make progress. Maybe it's just personal preference, but I'm a little bit allergic to salesmanship, so I don't like it when I read a paper, and I'm like, I don't know, man. I think you're sweeping some stuff under the rug. Essentially, you do not want me to be your reviewer number two.

Yeah. OK. So we'd like to avoid forgetting the general representation, and here's some engineering hacks that you can take if you want to try and do that. But then there's also this other kind of area of research that's specifically-- it's called domain adaptation. And the idea here is that maybe you have some source domain, and you have a target domain, and then you want to learn to do well on your target domain, but you also want to essentially make sure that the representation spaces of the source and target domain are well aligned.

And this shows up a lot when you have these types of distribution shift problems that we talked about before. But it also can be used for different types of fine-tuning. Essentially, one of the motivations here is that maybe if you force these feature spaces to be well aligned generally, that you can learn usefully how to map-- learn that mapping with much less data.

And one way that people do this is they actually-- they learn how to get the model from the source domain to the outputs, and then they try to learn a projection or a mapping from the target domain to the source domain. So you basically are learning how to map into the domain space where you had a lot of data, and then you can use the same model to actually do your prediction.

And there's lots of different ways to try to force these representation spaces to be aligned, but one kind of large set of them takes an adversarial approach. So here, you have both source and target data during training. And while you're training for every input, you want to predict the category you care about, and you also want to predict the domain that it came from, something about the distribution.

So a classic example of a data set in this domain adaptation space is the data set of different types of technological hardware. But one data set is from Amazon, and one is from eBay. So you can think that, OK, the way that people take pictures for Amazon are very nice and clean, and then the way people are taking pictures for eBay is maybe like on their messy desk. And so there, what you're trying to do is you're trying to predict, what kind of computer is it? And is it from Amazon or eBay?

And then this is where it gets adversarial. What they do is they flip the gradient from the domain predictor. So you're essentially trying to train the model to be as good as possible at predicting the category, while being as bad as possible at predicting the domain, which tries to force, essentially, the representations across the two domains to align.

And then the interesting thing is you can extend this to where you might not actually have category data for the source and the target. So maybe you had a bunch of category data for the source, and now, all you have for the target domain is just the knowledge of what domain it came from.

You can think about trying to still keep really good at predicting the category, but be really bad at predicting which domain it's from. And the hope is that there, even if you only have a very limited amount of labeled data in your target, or even sometimes none, you might still be able to learn something useful that helps you find the right categories in that target domain.

And then this is kind of expanded into this whole area of unsupervised domain adaptation, where here the idea is you might have data, but no labels from the target domain. So now, it's like you want to make a model that's as good as possible on some set of data, but you don't necessarily have any training labels. Now, why would we have that? So here's an example, actually from our own research, where we have a bunch of labeled source data.

In this case, the labeled source data is sonar images or sonar video from rivers with fish. And we're trying to count all the fish that are swimming upstream so we can make good sustainable fishing policies. And we have some rivers where we have a bunch of labeled data, but we want this to work on any river. Or if something changes about the river or they move the sensor, we want it to still work, even though there's that distribution shift. And so here, what we're trying to do is learn from labeled source data and unlabeled target data how get a model to work better on the target.

So how have people done this in the past? Well, so the very, very simple version of this would be, you take your target data set, you run it through a model that's been pretained on source, and then you maybe just calibrate it. You pick some threshold where you say, OK. Any prediction over this, we're going to treat as ground truth. So that's one way to transfer knowledge. You just use a model from a different domain and hope it works. It turns out it doesn't usually work very well.

So this kind of motivated this idea-- and this is a mechanism to do distillation, which we'll talk about a little bit more later. But what if, instead, we make a new model, that we're going to call a student model, and we're going to use those predictions on the target data as weak supervision?

So we'll use the predictions that our model trained on the source data gives, and then we'll try to train a model to predict the same thing. And the idea here is that maybe your student model is able to learned something useful that is better than just taking those raw predictions. And this does work. You get small gains, sometimes, over just the original source predictions. But this threshold is really, really finicky and hard to calibrate, right?

So people started adding in other bells and whistles. One thing is you use EMA, an Exponential Moving Average, to try to regularize the way that the weights change within the student and teacher model, so you let the teacher model itself also update slower. And then, finally, you can also introduce additional training on the source data. So there, maybe you're letting your student model learn from the source data and from the target data.

And then you can add in a bunch of fancy stuff, like instead of explicitly taking the predictions with a threshold, you can use a softmax. You can use a distillation loss so that you're learning to, instead of match the prediction, match the distribution over the outputs. And you can try to use some explicit feature alignment between source and target, so that's like basically trying to make sure that the features are overlapping as much as possible. Lots of fancy tricks here. Kind of complicated.

We essentially wanted to point out that these components are essentially the main components that people use to do this type of distribution, unsupervised domain adaptation. And the point here is that basically every modern domain adaptation architecture is some combination of these components. And so if you can centralize and unify the ways that people are actually doing these experiments, what we found is that you can often be more optimal than any previous approach.

And so these methods, many of them can be quite effective in certain scenarios. Of course, certain scenarios is a big interesting thing there. They're almost never as effective as having labeled data for your source domain. So if you just have labels, that's better. But if you don't have labels, you can actually start to get pretty close to performance if you do have labeled data if you build these things carefully.

And some intuition as to why that way might be is that, essentially, the model is able to learn something about the statistics of the target domain that's useful. You're shifting the weights towards being representative on those new statistics. Yeah?

AUDIENCE: How can you measure how the performance of this when deployed-- if you deploy it in a field that you actually don't have the labels of anything, how do you know your models did well?

SARA BEERY: Yeah. This is a really great point. I think it's actually the elephant in the room for most of the domain adaptation literature is that they almost always are demonstrating performance in their bold number in their table on labeled validation data for the target domain. But this kind of begs the question, if you have to label data to do model selection and determine what mechanism works best, why don't you just use that labeled data for training?

Yeah. I mean, I think it's kind of one of these interesting questions. And also, if you did have some training data, how does that kind of fit into this system? But yeah, I do think that in the literature in this space, one of the big missing pieces is that you still aren't very good at doing this if you don't have anything to select the best model or calibrate it in any way at all. Cool. Yeah?

AUDIENCE: I guess following up on that, I mean, isn't the point to reduce costs and time because you can use bigger data sets that are available?

SARA BEERY: So reducing costs and time are is valuable here. So maybe one dimension is you're trying to reduce the amount of human input you need to get a model to work well. But I think probably a better motivated scenario for a lot of these is places where we do not have the right bandwidth to be able to actually get the data to where a human could look at it easily.

So we use these domain adaptation models on these rivers that are in the remote Arctic, where the way for a human to get there is fly into Anchorage, take a seaplane, hike. It's just nontrivial to actually get to the sensor, and we don't have the kind of satellite connectivity or power that we would need to get the data to the cloud. So if you have these strong resource constraints on your system where you can't necessarily just have a human easily label data, I think that's where these types of models tend to make more sense.

Similarly, if you're-- I don't know-- doing navigation on the moon or deep sea, sometimes, we just need models that hopefully can adapt to a distribution. But the problem is if you don't have any kind of quality control going in, anything like this that's kind of learning from data where we don't have any of true labels, has the potential, from a reinforcement learning perspective, to veer off in some catastrophic way. And so you can fall into these kind of cycles where the model kind of gets worse and worse and worse and worse.

And then the other question is, how much source data do you still have to have access to? So I think essentially, there's a lot of engineering components to why you would want to do this and where it's actually effective and what the value add is. Yeah.

But I think there's also an interesting dimension here, which is these types of domain adaptation approaches also really benefit from the right starting point. So you can almost think of this as a different mechanism for fine-tuning, but the performance of these is very, very heavily influenced by the performance of the starting point model.

So it's almost complementary in terms of trying to build the most generalizable model, the most useful general representation. And then this is maybe a mechanism to fine-tune if you have a lot of data that doesn't have labels, instead of maybe a little bit of data that does have labels.

Cool. So next, I'm going to talk a little bit about how we might try to transfer information or knowledge about the outputs of a model. So if you recall before, we talked about how you train a classifier with something like cross entropy. So here, we have some model.

We have an input, x. We have some output, y, which is the ground truth for that model-- or y is the output prediction for that model. So maybe there's a bunch of different categories, and the one that has the largest prediction score or logit score is cat. And maybe, it's taken a softmax, actually, because it's going from 0 to 1, so this is maybe our prediction score.

And then the ground truth is going to be some one hot encoded label, where we say, all right. The true label for this input, x, is cat. And then we use cross entropy loss that basically tries to get our model to predict a distribution that's as close as possible to this ground truth distribution, which is just 1 for cat and 0 for everything else.

So you could argue that this is also kind of a teacher and a student. So here, the teacher is the difference between the ground truth and our prediction, and the student is kind of the way that we're capturing that distance. And so the idea of knowledge distillation is essentially, now, we have some big pretained model. We're going to assume that model is fixed.

So we have this little lock on here, so it's a locked model. And then we want to learn some student where-- so now, we go from x to y teacher, and we have a model that goes from x to y student. And then the way that we try to get this model to distill is we want the difference or the distance between the teacher model's predictions and the student's model's predictions to be as close as possible.

So now, maybe your teacher model has this distribution, and maybe it's predicting that cat is quite high, but it's also predicting that tiger is quite high. So now, when we're actually training our student model, we want it to similarly match that distribution. Now, does anyone have any intuition as to why this might be useful, why it might be useful to train a model to match a distribution of predictions as opposed to just a ground truth? Yeah.

AUDIENCE: Maybe the new model is smaller and quicker.

SARA BEERY: Yeah. So maybe it's a way to get a new model that's smaller, can be more computationally efficient, and can learn the same thing. Anyone have any intuition about why actually it might be valuable to match the distribution of the outputs? Maybe someone new. Yeah.

AUDIENCE: If there's a cat and a tiger [INAUDIBLE] example are kind of similar.

SARA BEERY: Yeah, exactly.

AUDIENCE: [INAUDIBLE] animals. Maybe a dog's more similar than the monkey. [INAUDIBLE] a little bit to the dog. It'll give it some more information.

SARA BEERY: Yeah, so that's exactly the answer I was looking for. It's giving the model some information about the similarity. So here, cats and tigers are maybe similar, but it's not just the similarity of the category set, it's the similarity of what's recognizable in a given input.

So if you have an image of a cat, but it's an orange cat, and it's in tall grass, you could see how a model would say, hey, that's pretty similar to a lot of the data I've seen of tigers. And so now, what you're teaching the model is not just, this is a cat, you're teaching it, this is a cat that looks a little bit like a tiger. And that might actually be useful context for a model in terms of learning more than just a set of separable categories.

And so there's another kind of interesting point here that I think is important to raise that's not just about efficiency but is about the modern era of many closed source models.

So if you have access to the outputs of a model, its predictive scores, but not the model itself, you could arguably learn a model that can match the predictive scores of a given output. And this is maybe one of the reasons that a lot of those closed source models don't actually give you output predictions over the whole category set.

Cool. So now, you can essentially think about doing standard cross entropy, but instead of cross entropy being to that binary category, it's just between the teacher and the student outputs.

And so exactly like what you said before-- you guys are great here-- the label you get for every input might be more informative than just the ground truth class, and so this soft target is telling you something about the similarity and what things look like. So this is a fun example. It's not just saying this is a dog, but it's saying, this is a dog that looks a little bit like a cat, and it looks very different from a bear. And so maybe that's telling you oh, it's a small dog. It's-- I don't know-- one of those little yappy dogs.

So this original introductory paper, which is one of Jeff Hinton's, shows that you might actually get lower training accuracy when training this way, but you end up getting higher test accuracy on this speech recognition task. And so the other interesting thing is if these soft targets have really high entropy, they're giving a lot more information per training case than hard targets.

And they have a lot less variance in the gradient between training cases-- is another thing that they showed in this paper. And so a small model can often be trained on a lot less data than the original model while using a much higher learning rate when you're doing this type of distillation.

But it's not just something you can use to make a smaller model. You can also think about learning from a distillation perspective across modalities. So here, maybe you have a teacher that learns to predict categories from an RGB image, and then you have a student that only has access to depth. So this might be useful if you have labeled data for RGB, but you don't necessarily have labeled data for depth.

And so another example of this cross-modal distillation is this model called SoundNet where they have waveforms and unlabeled video, and then they essentially are doing object distributions and scene distributions, and then they're learning distillation across both.

There's another interesting version of distillation from a computational complexity perspective where you can actually distill the results of an ensemble. So often, we find that, even in the modern era, ensembles over lots of models are often more predictive than any single model. One of the arguments for this is kind of a classical one, which is that every model is going to hopefully make the same correct answers more often than they make the same mistakes.

And so here, what you can think about is maybe you have an ensemble that's really great ensemble, an ensemble of three different models, but they're all kind of expensive to run, and you don't really want to be running three models all the time. So you can think about explicitly taking that ensemble of model outputs and then trying to distill a model that matches the distribution of your ensemble. And again, each model is imperfect in different random ways.

Errors cancel each other out, and the truth is shared. Of course, this isn't always true, because sometimes, we have biases in models that are learned from our data that will be consistent for every model. So the thing you're not going to get out of this is something that's really good at the types of corner cases that every single model in your ensemble gets incorrect. But it can be much more efficient.

Cool. So what if you don't just want to distill the outputs, but you also want to capture and distill something about the knowledge of the representation itself? So you might want to, for example, match the embeddings at some intermediate point within your model using contrastive training. So as opposed to just matching the logits or the output scores, you can actually think about distillation specifically using contrastive learning but where you want to actually supervise embeddings internal to the model to be kind of invariant to some viewing transformation, for example.

So that viewing transformation-- we talked about rotation invariance, et cetera, before-- but another version of viewing transformation could be the different views of that representation from the teacher, which is a big network, versus the student, which is a small network. Yeah?

AUDIENCE: So in practice, when you see this and what you were talking about earlier, how much smaller than the teacher is the student model?

SARA BEERY: Yeah. That's an interesting question. In practice, how much smaller than the teacher is the student? So again, in my experience, it kind of depends on the data that you're trying to get the model to work well on and how complex it is, how complex the task is. So how small you can get a model to be and still be useful often is really directly related to the complexity of the information you're trying to model.

And there's a lot of different ways that people try to get small models. If you want a machine learning model to work on your phone, it just has to be smaller. You're not going to put it on an H200-- an H200 scale model on your phone. But so people use distillation to make small models, but they also have a lot of other mechanisms for getting models to be small, things like model compression or quantization, for example, that are doing things-- or pruning, even just explicitly removing weights from a model-- and trying to get the model to work pretty well.

And then a lot of times, methods that work quite well in terms of getting really small models to be really good are some combination of all of the above. So do some pruning, maybe do some distillation, et cetera. Yeah.

AUDIENCE: Is that basically related to how we see cloud Gemini that have different sizes of model-- is that the result of the techniques you talk about?

SARA BEERY: So you were asking about if you see these different sizes of models from some of these foundation models-- for example, the Gemini model, or some of the OpenAI, like ChatGPT models-- why they have different sizes and whether it's actually using distillation versus just training those models from scratch at that size. They don't tell me their trade secrets, so I don't know.

But I will say the reason that you see these different models put out versions that are small versus big is because they know that different downstream use cases will have access to different resources. Often, it's kind of like-- the big models are better than the small models, but you use the biggest model you can afford where afford might be a physical constraint, it might be a monetary constraint, speed. Cool.

Anyway, cool. So here, instead of thinking about rotations or whatever, the two different views that we want to be similar in this contrastive way are maybe intermediate representations from two different models, and then the idea is that you try to get this student to output embeddings or output a representation for input data that's similar to the teacher's representation.

And there's this cool thing that you can see in the logit space. So specifically, if you train these-- do this representation distillation, and you look at the correlation coefficients between the class output logits-- so essentially, the student's logits minus the teacher's logits-- on CIFAR10, these hundred classes by hundred classes-- you can see that you get this much, much cleaner behavior when you do this kind of contrastive representation distillation versus just doing standard knowledge distillation.

Cool. So talking again about speed, compression versus distillation. So there are a lot of different mechanisms for trying to make models small. So both compression and distillation will give you smaller models with similar accuracy to large ones. Distillation will use SGD to train a smaller model to match the characteristics of a large one, so stochastic gradient descent.

That process of training can be computationally expensive. Compression, using maybe pruning or quantization to try to reduce the computational cost of a large model, that can be much more efficient from a computational perspective. So sometimes, people want to use compression over distillation just because of the speed or the computational cost of training.

So here is an example from a pretty classic work called optimal brain compression that's essentially doing both quantization and pruning. And here, they're kind of getting a bit of the best of both worlds. So pruning is super computationally efficient because you're just deleting stuff.

Quantization can be a little bit more expensive and can also explicitly have some components that feel similar to distillation because you're basically trying to say, I want intermediate representations to be similar. And so here, they might come up with some of interesting combination of a mask where you're kind of getting efficient, global optimal compression in some way, but without it actually being computationally expensive.

Why do you want compression that isn't computationally expensive? Again, because it's cool if you can actually directly compress a model on something like a device. So maybe you don't want to have to do the compression on the cloud. You actually need to do the compression on the edge.

And then you can actually get things to be even smaller and maintain accuracy if you let them specialize to a specific distribution while you're compressing them. So maybe you have a large scale generalist model, and what you need is a small scale specialist model-- so for example, a model that only knows about sharks.

We were able to show that you can really increase the quantization, the amount that you can remove, and maintain accuracy if you're letting the model specialize as well. And that seems pretty intuitive. If you don't need to keep as much of the original information, you can make the model much smaller. Cool.

So towards the end, we're going to dig into foundation models. So foundation models is a term that was coined by the computer science department of Stanford.

And essentially, it's just trying to make a semantic concept for this idea of really useful transfer learning models, the idea that you have some models that are so generally useful that they end up being really effective for a bunch of different downstream tasks.

And I think you can put different foundation models in different buckets, I would say. There's some foundation models that are fundamentally representation learning models where then the question is, essentially, do you need to fine-tune at all to make the model useful?

And I would say as foundation models become more and more general, you need to fine-tune them less and less, because maybe the set of things that you are interested in is more likely to be within the very, very broad distribution that's covered by that original model. And mostly, if a model doesn't need to be fine-tuned, it becomes more accessible.

So this is kind of the power of something like ChatGPT or something like CLIP or something like some of these large scale foundational generative models, for example, like image generation, is that to be useful for any given end user for the vast majority of things that people are kind of interested in, they don't need to be retrained.

They're kind of usable off the shelf. And maybe one way you could think about foundation models is like their use is directly related to what percent of the things people want to do, this model can do off the shelf without any of specialized training.

Cool. And I like this quote. I think this either came from Phil or Bill. "If I have seen further, it is by standing on the shoulders of Giants." It's a quote by Newton. And this is a fun painting. *Blind Orion Searching for the Rising Sun* by Poussin. But I think the idea here is that the bigger the foundation, maybe the further up we get as a starting point.

So what does this really mean? Well, so this is a plot from OpenAI, and it's only going up to 2018. I should get a newer version of this that captures how big these models have gotten since. But it's really just tracking the fact that-- note that this is on a log scale, so we're really seeing exponential growth in the size of models over time.

This is the amount of compute. So going from AlexNet to the AlphaGo model back in 2018 was a 300,000 times increase in compute. And now, some of the models we use are in the billions and trillions of parameters. So again, much, much more. And these models tend to be quite generalizable. These very large models are much more generally useful.

I think there's some interesting kind of dimensions here. So one thing is if you look at just computing power usage over time-- so this one goes up to 2021-- you can see that academia and even academia industry collaboration, that computational power use is staying about the same. But industry, that's starting to scale pretty significantly. So you could consider this a, who is building these bigger and bigger models? And definitely, I think industry is a-- well, they're often the only people who can afford it.

And I do just want to raise-- because it's something that I feel really strongly about-- that these foundation models aren't free. So they're definitely not free from the sense of-- the companies are paying a ton in compute and in power and in water usage to be able to train them.

Microsoft, I think, announced that they're spending $80 billion next year on compute. So that's a lot of money. That's not something that nonindustry is going to be able to keep up with. A lot of that is that Microsoft is actually the ones who are running a lot of the data centers that then other groups like OpenAI use.

But these also have a cost for us as a society in terms of their climate cost. So this is from a paper that was looking at estimating the carbon footprint of one 176 billion parameter language model from Microsoft. And they were looking at the entire-- they were saying, OK.

If we go from the entire scope of the climate cost of this model, that would include things like raw material extraction, so mining, the manufacturing costs of the materials, the equipment manufacturing, model training, model deployment, and disposal and end of life of all of those hardware components-- they were only focused on these three in the middle, so manufacturing, model training, and model deployment.

And what they found was a huge factor, specifically in the power consumption and how that relates to carbon emissions, is the sustainability of the data centers' energy sources. So even much smaller models can have a much, much higher carbon emission if they're being trained on data centers that are not renewable power.

And I actually think that one of the benefits of this compute boom has been the increased focus on renewable energy and sustainable power within data centers from these companies because it's really well motivated by their bottom line. They're paying a lot of money for this. But the climate cost of these emissions is something that they are not necessarily going to foot the bill for.

And to put this into scale, I mean, I think power usage and electricity is one big dimension, but I think water use is another really important dimension. I think it was GPT-3-- was trained mostly at data centers in the state of Iowa. And the water use in the state went up by 700% while they were training the models. I read an estimate recently that said that about 50 queries to ChatGPT is a bottle of water.

And so if we think about, now, resource access globally, we already know that we are running out of water. And now, having these models-- and who is benefiting from them being successful? And then what that means about global society and access to really necessary resources like water is complicated. Yeah.

AUDIENCE: Just for people who may be not aware, is the water consumption like--

SARA BEERY: Cooling.

AUDIENCE: Cooling. And then is that water-- let's say-- I assume they're not using-- although maybe they are-- potable water on the way in. Does it become unpotable when you use it for cooling? It sounds like it's bad. You're not getting the water back.

SARA BEERY: No. Usually-- I mean, I think there's a lot of, actually, research now into, can we do this type of cooling with seawater, for example? Or what are the effects of running this water through these cooling systems on the portability of the water? I'm not going to speak to that because this is outside of the scope of my understanding of these mechanical engineering systems or thermal and hydro.

But I will say, I think that there's a booming cottage industry of researchers who are trying to figure out how to make data centers less climate intensive. But I'm also sure that all of you know that we are vastly outstripping our capacity in terms of things like power. It's been very big in the news that Google is considering reopening Three Mile Island to power data centers. So we're thinking about moving back to nuclear power just to keep up with the AI hunger of this.

And from an ethical dimension, it does get complicated, because the scale of resources being used versus who actually is benefiting is a bit messy. And I think a lot of groups or a lot of companies would have you believe that these foundation models are so beneficial to society that it's necessary. But I am less clear on that.

And one of the things that I want-- I want the option for a climate friendly search to come back. I don't want Google to run AI models every time I'm trying to Google something. I don't really need the description.

I would love to have an opt in that says no, I want to go back to MapReduce. That was much less computationally expensive. Can you please give me the dumber search that's more climate friendly? So I've been talking to people about that. Maybe we'll start to see it. Yeah.

AUDIENCE: Is there any-- on that same kind of line-- any kind of research on how much time, like a compute time, [INAUDIBLE]? When I use Google, I have to google 10 things before I get the answer. But ChatGPT, I ask it once, and it gives me all the information.

SARA BEERY: So the question is basically the time versus computational trade off if you query ChatGPT once versus Google searching 10 different things. I don't know. Yeah. I don't know. I think it depends also on how these things are architectured in terms of things like global similarity search. It also is, how good are you at using Google? Yeah.

The thing I would really love to see is more transparency about all of this from the side of companies. What actually is the computational cost relative of these different things? And a lot of companies are looking at, for example, making use of these different sizes of models and getting really creative about routing queries to the less expensive models instead of the bigger ones, unless they're more complex. How do you be efficient with it? But yeah, I think-- yeah. I think there's a lot of room to improve still.

Cool. So what does it take to train one of these models? These are, again, a little bit out of date. We have much bigger models now. But Dalle-2 required 650 million training examples, semi-curated text to image pairs. GPT-3 cost estimated around $4.6 million to train once. I'm sure you guys can imagine that we don't train these models once. We have to do some hyperparameter searching. There's like some differences in comparison. The PaLM model had 67 authors. That's 67 engineers working on this. The numbers are increasing.

But the argument, I guess, is these numbers might be small if we think of the result as foundational infrastructure. So if you compare this maybe to the cost of building an interstate highway system, or the cost of developing OS X, maybe there's different ways that we should be thinking about what the cost is if this is going to be fundamentally infrastructure that is foundational.

And one of the motivations for that perspective is that if we look at scaling, as you go up in these different sizes of models, and you look at the number of model parameters, and you look at some metric of performance on a bunch of different tasks-- so 58 different tasks-- what you start to see is that you really don't start to see model performance get significantly better, where maybe it's, quote, unquote, working as well as an average human on some set of these tasks, until you have much, much larger model sizes. My argument would be, OK. Now, let's do research into efficiency to get these back down.

So what does that actually mean if you're thinking about learning foundation models? So one, you have tons of data. You have some learner. And then you're generating all these different types of models that, in many cases, are kind of ready to use. So now, you want to use or adapt those foundations with little or no data to solve new problems.

So you now might have some adapter, which maybe could be something like an AI agent, or it could just be something that's quite simple, where-- OK. We just pull in CLIP. We have little to no data. You use your adapter, and you generate some app. And I would argue that this is a lot of the most recent set of startups in Silicon Valley have been some version of this, this "just ask" mentality, where it's like, OK.

We take one of these large language models, and then we come up with a really clever way to ask it to do stuff for us. And it's not really machine learning anymore. It then turns into this kind of new paradigm of research that people call prompt engineering.

So what does this maybe look like? So here, you have something like GPT-3 at a very, very surface level. It's trained autoregressively to predict the next character, so colorless green ideas sleep, furiously. So now, the adapter, again, becomes this idea of prompt engineering. So you say, OK. Here's a review. Now, you tell it to fill in the blanks. The sentiment in this review is blank. And it will fill in the blanks and tell you positive.

And I'm sure many of us have played with the capabilities of these models. They are shockingly good. They get better all the time. You can ask it dumb stuff like, what are the top 10 dishes at a specific restaurant in Cambridge? And it will give you a list of dishes and rank them based on who knows what?

If you say, OK. Here's a list of primary numbers, and ask it to continue, it will continue that list of primary numbers. This is where it gets interesting, because depending on who you talk to, it's either learning complex patterns and learning to mathematically reproduce them, or it's learning to do pattern matching. It's essentially just a kernel machine, and you've just given it everything it's potentially ever going to see in terms of its training data.

And you can also think about what people call in-context learning. So here, that kind of outer loop, which is like learning via stochastic gradient descent during supervised pretaining, then the inner loop is just this in-context learning where you essentially say, here's some examples of what I'd like you to do, and then ask it to continue to do them. And this kind of starts to feel flavored, like meta-learning, which we'll talk about a bit more on Tuesday.

And you can also think about its ability to extrapolate. So here, you show a bunch of elephants that are counting up, its icons, or whatever. And then you ask it, OK. Starting with these four cases, count, and it will do so. Or it can do things like translate and translate just from a few examples of English and Spanish. But again, what is the model actually doing under the hood? Is it learning something, or is it finding the right kind of pattern to match within its training data?

So what does it actually do? How do we actually do good prompting? It turns out that handcrafting good prompts can be really tricky, and these models are incredibly brittle to prompting. So you can give it a different template and ask it very simple things. And the behavior of the system will be really sensitive to how the question is asked.

And there's a funny-- someone said that it was a little bit like working with a genie in a lamp. You're like, if you asked the genie for no pain on Earth, it might kill everybody. You have to be very careful about doing this type of prompting. And then one of the things that people do is they think about ensembling. OK, we'll prompt the model with a bunch of different things, and then we'll ensemble the predictions. Of course, that's really computationally expensive.

So there's a bunch of research now into, how do we actually prompt these foundation models effectively? One is this idea of soft prompting, which basically looks at trying to optimize some of small continuous task-specific vector, like encoding, while the language or the model parameters are frozen. So here, it's like you have some objective. It's now kind of back to the normal learning framework, but we're essentially just trying to find the optimal words to use as the input to the model.

And so this paper specifically looks at fine-tuning these input embeddings, these little specializer tokens, that get prepended to your input task, and then they help you optimize the result for some task at hand. So the tokens might be somewhat gibberish, but they're really good at finding the place in the distribution, or helping the model do that pattern matching so that it gets you close to something that's effective.

There's also this idea of standard prompting, which is like, here's a question and an answer, and then here's a new question. And then you ask the model to answer. And then what chain-of-thought prompting does is it essentially demonstrates an answer that has reasoning in it. So you basically tell the model that when you answer, you want it to answer with logic, and it's more likely that it will give you an answer that has some logic.

And then it's been shown that the answer is more likely to be right. There's also really dumb stuff you can do, which is like, OK, you are very smart. And then ask it a question. Tell the model, you're a scientist. Now, tell me-- which is really funny. And it's one of the reasons that people start anthropomorphizing these things. It's like, why do you need to give it compliments? But actually, I think it's really about finding the right part of the distribution.

You want to find the part of the distribution that's learned from text, that's maybe from scientists, and maybe more likely to be correct. Anyway, this type of chain-of-thought prompting, we're telling it how to answer, was shown to be better using than standard prompting by a lot and better than the prior best on these types of math word problems.

And then CLIP is also a foundation model. And so this is contrastive learning where you're learning semantic similarity between text and images. And so one thing you could do is you could treat this CLIP model as a representation learning model. So the adapter could be a linear classifier built on top of this image encodings. And they have shown that you can do that. So this is the number of labeled images per category.

But you can also think of it as an adapter where, now, you just ask. So instead of saying, let's do some fine-tuning over the set of categories I care about, you can just say, find me the text that is most similar to this image that's of the form a photo of a-- whatever. And this is where this stuff gets really weird.

Because if you say, OK, you have the text embedding of a photo of a dog versus the text embedding of photo of dog, those will be different enough that it will massively change the performance of this kind of zero-shot "just ask" categorization on CLIP. So they are very, very brittle.

But you can also now think about, OK. I'm going to encode a sketch of a banana, and I'm going to encode a photo of a banana. And I'm going to show it this. And I'm going to say, which one has the closest cosine similarity in that joint text image embedding space? And that will tell me which one it matches more closely. So this is more like a sketch of a banana than it is like a photo of banana.

One of the interesting things-- we've been working on some of this text-based query for large image databases for iNaturalist, which is a really large photo repository of species data. And these CLIP models are very good at simple concepts, but they're really bad at composition. So they're good at ors, and they're bad at ands, if that makes sense.

And one of the reasons for that is you can think of these encodings as almost a bit like a bag of words. So if you say, I want to find photos of a juvenile bald eagle, it will find photos of juvenile things and photos of bald eagles. But it's not necessarily going to prioritize photos that are juvenile and bald eagle.

So that type of compositional querying is a whole other area of research trying to make these models better at compositions. And there was a really fun paper in the last year or so that was teaching CLIP to say no. This is also really bad at-- if I say not a photo of a bald eagle, it would just find a bunch of photos of bald eagles. It's bad at the negation.

Cool. So can you think about prompting in other modalities, too, which is actually kind of fun. It's just a bit of a psychological trick. So here's this idea of CLIP prompting. So instead of modifying the text encoder, saying, OK, here's some different possible text descriptions, like finding matches, you can also look at mechanisms to modify the image encoder with a visual prompt. So here, this is an example where they've learned this optimal kind of border thing that they put on the image.

And it's kind of similar to adversarial noise. We learned about learning adversarial noise, but this is now like learning beneficial noise. It's like, what's the thing that you can add to your image that will maximize the likelihood that this CLIP model will properly map the correct category? Which is fascinating. So given a fixed model, you can learn from a data set, what can you add? What types of noise can you add to the image to actually make the model better at recognizing dogs versus other things?

And this is noise that helps, no matter what the input image is, for some fixed data set, domain, or paradigm. So you have some training data, and you're basically finding some constrained set of noise where, if you add that noise to any image, it maximally improves the prediction for the correct class. Kind of trippy, right? It's cool. But really, what it's saying is something weird about the underlying statistics of the model itself.

So here, it's almost like you're adversarially reprogramming the model, so you can also think about learning them in an adversarial way where it's like, you find this thing you can add to images that will make it maximally wrong in this CLIP embedding space.

And this is a really weird version of this. So here, this was an experiment where they basically took a bunch of satellite imagery, and they found that adding a single bright pink pixel-- I don't know if you guys can see it. It's up there, this one little pink pixel. They found that if they add a single pink pixel to every single satellite image in the same place, it improves performance by a lot.

AUDIENCE: Is that-- I mean, did they check if, in the training data set for this model, it statistically was-- I guess you're saying it's for every class.

SARA BEERY: For every class. But I mean, if you have a fixed model and a fixed data set, then you can figure out, what is-- in this particular constraint, I think they were saying, we want to add a single pixel-- we want to change a single pixel value to a specific color, find the optimal version. And weirdly, it gives a 3% boost in accuracy across all the categories.

Anyway. So I think that this type of work-- this is not saying the best thing to do to make these models work really well is do one of these weird tricks like this. I think it's more enlightening about what the heck these models are learning, how fragile and complex the statistical representations are.

And you can think about this adaptation happening on the inputs, on the mapping, or on the outputs, this type of prompt-based adaptation. So you can take these pretained models, and you can do fine-tuning. You can take pretained models, and you can just learn some linear layer that does some reprojection. Or you can try and do this type of visual prompting where you're explicitly adding something to the top of an image, and then now, it's getting a better prediction.

And there's a bunch of interesting stuff where people show that you can do this visual prompt tuning for these really large models that can be useful even when the model weights or the logits are not available. It's kind of like distillation. If you just add things to the input, see what the output is, you don't need access to the underlying model to find this type of visual prompting that's valuable.

And then there's another fun version of this, which is like fill in the blank style tuning. So basically, these people figured out how to get an image foundation model to do segmentation via in-context prompting. So they made these little grids of images where they would show an image with an example of a segmentation mask for a class, and then another image, and then have this empty area.

And then they would use an inpainting foundation model to get it to prompt you to do segmentation, even without being explicitly trained to do segmentation. And then they showed this for many different types of things. So edge detection, colorization, inpainting. You're essentially asking these models-- you're prompting them with, here's an example of what I want with one case, and then it's just giving you the kind of correct label across a bunch of different possible downstream types of categories.

So these foundation models really have a lot of capacity, and there's so many creative ways that we can try to harness that capacity that I think people have looked at. And there's some interesting resources if you guys are interested.

So in the last minute-- sorry. We're kind of running late today. Combining foundation models. So you can learn foundation models and adapt them, but you can also think about taking these combinations of foundation models and building things that are kind of nonmodular on top of, again, these building blocks. So you want to do something that's text to image. You can take VQGAN and CLIP.

You can plug pretained models together. I think we showed this early on. And so this is really just thinking about plugging these different image foundation models together. You have some image generator. You have a CLIP model that's got this similarity foundations and semantic meaning between text and images.

And then you can do something optimize this little embedding vector, this latent space vector, to get an image that maximally matches that text prompt. And there's a bunch of recent work on this text-based image generation. And Dalle-2 is a good example of that, where, essentially, they have a CLIP objective, and then they also have this prior that helps them do decoding slightly more efficiently.

Cool. So today, we talked about learning with little data, transferring knowledge, about the mapping, the outputs, and a bit about foundation models. We'll follow up on Tuesday. And don't be too sad, but think twice about ChatGPT.