The Future of AI

Megha Sagar Patil
Department of BCA, Vivekanand College, Kolhapur,Maharashtra, India

Sairaj Satish Suryavanshi
Department of BCA, Vivekanand College, Kolhapur,Maharashtra, India

Satyajeet Sanjay Mandale
Department of BCA, Vivekanand College, Kolhapur,Maharashtra, India

So when I first started teaching this class, uh, seven years ago, I used to have to motivate why AI was important and why if you study it you'll have a lot of impact in the world. But I feel like I don't really need to do this. Now it's kind of inescapable that you pick up the news in the morning and you hear something about, you know, AI. And indeed we've seen a lot of success stories, right? AIs that can play Jeopardy or play Go, Dota 2, pro- even poker, all these kind of games at super human level performance. It can also, you know, read documents and answer questions, do speech recognition, uh, face recognition, um, even kind of medical imaging. And all these tasks are, uh, you read about how successful these, uh, technologies have been. Um, and then if you take a look at outside the kind of the technical circles, there's a lot of people, um, in policy, um, and trying to ask what is going on with AI. And you, you hear about, uh, these kind of very, uh, broad claims of how transformative AI will be, um, to the future of work and, um, to society and so on, and even some kind of bordering on, uh, pretty castro- you know, catastrophic consequences. So what's gonna happen in the future, no one knows, but it is fair to say that AI will be transformative. Um, but how do we get here? And to do that, I wanna take a step back to the summer of 1956. So the place was Dartmouth College, John McCarthy, who was then at MIT, and then, uh, after that he founded the Stanford AI Lab, um, organized a workshop at Dartmouth College with, um, some of the best and brightest minds of the time; Marvin Minsky, Claude Shannon, and so on. And they had this not so modest goal of trying to think that every aspect of learning or any feature of intelligence could be precisely captured so that a machine can be just, uh, simulated.

So they were after the, the big question of how do you kind of solve, um, AI. So now they didn't make that much, uh, progress over the, the summer, but a lot of programs and interesting artifacts came about from that time. Um, there were programs that could play checkers or prove, uh, theorems, and sometimes even better than what, um, you know, the human proof will look like. Um, and there was a lot of optimism. People were really, really excited, and you can see these quotes by all these excited people who proclaimed that AI would be solved in a matter of years. But we know that didn't really happen and there's this kind of folklore example, um, people are trying to do machine translation. So you take an

English sentence like 'The spirit is willing but the flesh is weak', you translate into Russian, which is what, um, the choice language by the US government was at that time, and you could, uh, translate back into English; and this is what you get, 'The vodka is good but the meat is rotten'. Um, so the government didn't think that was too funny, so they cut off the funding and, um, it became the first AI winter. Um, so, so there was a period where, you know, AI research was not very active and was not well- very well funded. Um, so what went wrong here? Um, these were really smart people, right? Um, they just got a little maybe ahead of themselves. So two problems; one is that the compute was simply not there, right? It was millions or even billions of order of magnitude compared less than what we have, uh, right now. And also, the problems, the way they formulate them, intrinsically relied on camp- exponential search which, um, no matter how much compute you have, you're never going to, you know, um, win that race. Um, they also have limited, you know, information, and this is maybe a kind of a more subtle point that if I gave you infinite compute and I asked you to translate, I don't think you would be able to figure it out because it's not a computation problem.

You just need to learn the language and you need to experience all the subtleties of language to be able to, you know, translate . But on the other hand, AI wasn't solved, but a lot of interesting, um, contributions to computer science came out of it. Lisp was- is- uh, had a lot of ideas that underlay ma- many of the high level programming languages we have, garbage collection, um, time-sharing, allowing, uh, multiple people to use the same- one computer at the same time, which is something that, uh, we kind of take for granted. And also this paradigm of separating what you want to compute, which is modeling, and how you do it, which is inference, which we'll get to a little bit later. Okay. So um, people forget quickly and, um, in the '70s and '80s, there was a renewed generation of people getting excited about AI again. Um, and this time it was all about knowledge, right? Knowledge is power and, um, there were a lot of expert systems which were created. And the idea is that if you could encode expert's knowledge about the world, then you could do kind of amazing things, and at the time the knowledge was encoded in generally a set of rules. Um, and there were a lot of programs that was written, and you'll notice that the, the scope is much narrower now. The goal isn't to solve it- all of AI, but to really focus on some choice and problems like diagnosing the diseases or converting customer's order parts into parts, and, uh- customer orders into parts and, uh, this was the first time that AI, I think, really had a real impact on industries. So uh, people were actually able to make useful, you know, products out of this. And knowledge did actually play a key ingredient in curbing this, you know, exponential growth that people were worried about. But of course, um, it didn't last long. Um, knowledge as deterministic rules was simply not rich enough to capture all the kind of nuances of the world. It required a lot of manual effort to maintain and, um, again, um, a pattern of over-promising and under-delivering that seems to plague, um, AI people, led to the collapse of the field and the kind of a second AI winter.

Um, okay, so that's not the end of the story either. But actually it's not kind of really the beginning either. Um, so I'm going to step back further in time to 1943. So what happened in 1943? So there was, um, a neuroscientist, McCulloch; and logician, Pitts, who were wondering and marveling at how the human brain is able to do all of these kind of

complicated things. And they wanted to kind of formulate a theory about how this could all happen. So they developed a theory of, um, artificial neural networks, um, and this is kind of you can think about the root as of, you know, deep learning in some sense. Um, and what's interesting is that they looked at, um, neurons and logic, which are two things that you might not kind of necessarily associate with each other, and showed how they were kind of connected mathematically. And a lot of that early work in this era were of- around artificial neural networks, was about studying them kinda from a mathematical perspective. Um, because at that time, the compute wasn't there, you couldn't really run any kind of training new models or um. And then 1969, something interesting happened. So there's this book by Minsky and Papert called Perceptrons. And this book did a lot of mathematical analysis. And it also showed that linear models, one of the results of many, was showing that linear classifiers couldn't solve the XOR problem. Um, the problem is- another way to think about the problem is basically given two inputs, can you tell whether they are the same or not, or different. And, um, so it's kind of not a- shouldn't be a hard problem but linear classifiers can do it. And for some reason, which I don't quite understand, it killed off neural nets research even though they had said nothing about if you had a deeper network, what it could do.

Um, but it's often cited that this book, ah, swung things from people who were interested in neural networks to the field of AI being very symbolic and logic driven. Um, but there was always this kinda minority group, um, who were really invested in and believed in, um, the power of neural networks, and I think this was always just kind of a matter of time. So in the '80s, there was a renewed interest. Um, people kind of discovered or rediscovered the backpropagation algorithm which allowed a kind of, for a generic algorithm that could train these multilayer neural networks because single layer remember was insufficient to do a lot of things. And then one of the kind of the early success stories, as Yann LeCun in 1989, applied a convolutional neural network and was able to recognize hand digit- written digits, and this actually got deployed, um, by the USPS and was reading kind of zip codes. Um, so this was, you know, great, ah, but it wasn't until this decade that the, um, this area of neural networks really kind of took off, um, under the moniker deep learning. Um, and, you know, AlexNet in 2012 was kind of a huge transformation, um, where they show gains on the, kind of ImageNet ba- benchmark and overnight transformed the computer vision community. Um, AlphaGo as, you know, many of you know, and many kind of other, um, and there were kind of the rest is history. Okay, so- so there's this kind of two intellectual traditions. Um, you know, the name AI has always been associated with the kind of John McCarthy logical tradition, that's kind of where it started. But, um, as you can see that there is also kind of this neuroscience inspired tradition of AI, and the two are kind of really had some deep philosophical differences and over the decades fought with each other kind of quite a bit. But I want to pause for a moment and really think about, maybe if there were actually kind of deeper connections here. Remember McCulloch and Pitts, they were studying artificial and neural networks, but the connection was to logic, right? So from even in the very beginning, there is kind of this synergy that, you know, some- some people can kind of often overlook.

And if you take a look at AlphaGo, which if you think about the game of Go or many games, it's a mathematically, you can write down the rules of Go in logic in just a few lines. So it's a mathematically well-defined logical- logic puzzle in some sense. But somehow, the- the power of neural networks allows you to develop these models that actually play Go really- really well. So this is kinda one of the deep mysteries that has, kind of, uh, I think is kind of o- opens standard challenge, you know, in AI. Um, as with any story it's not a full picture, and I want to point out on this slide that, AI has drawn from a lot of different, you know, fields, many of the techniques that we're gonna look at, for example, maximum likelihood, came from your statistics or games came from economics, optimizations, gradient descent, hence from- was, you know, in the '50s completely unrelated to AI. But these techniques kind of developed in a different context. And so AI is kind of like, you know, it's kind of like a New York City. It's- it's like a melting pot where a lot of the- these techniques that kind of unified and apply to kind of interesting problems. And that's what makes it, I think really interesting because of the- the new avenues that are opened up by kind of unique combinations of, um, existing techniques. Okay, so- so that was a really bre- brief history of, you know, where- how we got here. Um, now I want to pause for a moment and think about, you know, what is- what is the goal? What- what AI people are trying to do? And again this- this is kind of there's two ways to think about this which and- sometimes the conflation of these causes a lot of confusion. Um, so I like to think about it as AI as agents, and AI as tools. So the first view asks the kind of standard question of, how can we create or recreate intelligence? And the second one asked, you know, how can we use technology to kind of benefit, you know, society? And these two are obviously very related and they have, ah, a lot of shared technical, um, overlap, but, you know, philosophically they're kind of different.

So let me kind of explain this a little bit. So the idea with AI agents is, and this is, I think a lot of what, um, um, gets associated with AI, um, and especially as, you know, with science fiction. That kind of, ah, po- portrayal certainly kind of encourages this kinda view where you're human- we're human beings. And what you do is you look in the mirror and you say, wow, that's must- that's a really smart person. And you think okay, how- how- what- what- what can humans do that is, you know, so amazing. Well, they can, um, they can see and they can perceive the world, recognize objects. Um, they can grasp cups and drink water and not spill it. Um, they can communicate using language as I'm doing to you right now. Um, we know facts about the world, declarative knowledge such as what's the capital of France and procedural knowledge like how to ride a bike. We can reason with this knowledge and maybe ride a bike to the capital of France. And then, really importantly, we're not born with all of this, right? We're born with basically nothing, none of these capabilities, but we are born with the capacity and potential to acquire these over time through experience. And learning it seems to be kind of this critical ingredient, which drives a lot of the success in AI today but also with, um, you, know, human intelligence it's clear that learning plays such a central role in getting us to the level that we're operating at. So each of these areas has kind of spawned entire sub-fields, and people in it are kind of wondering about how you can make artificial systems that have the language, or the motor, or the visual perceptual capabilities that, you know, humans have.

But are we there yet? Um, and I would- I would like to think that we are, ah, very far. So if you look at the way that machines are, have been successful, it's all with a narrow set of tasks and, you know, millions or billions of examples, and you just crunch a lot of computation, and you can really kind of optimize, um, every- any tasks that you're going to come-come up with. Whereas humans operate in a very different regime. They don't necessarily do any, you know, one thing well, but they are have such a kind of diverse set of, you know, experiences, can solve a diverse set of tasks and learn from each individual tasks from very few examples. And still it's a kind of a grand challenge, in from a, uh, cognitive perspective, how you can build systems with this level of capability in that humans have. So the other view is, you know, AI tools. Basically we say okay well, you know, it's kind of cool to think about how we can, uh, you know, recreate intelligence. But, you know, we don't really care about making more, um, things like humans. We already have a way of, you know, doing that, that's called babies. . Um, so when instead what we'd really like to do is not making something that's like a human but making systems that help humans. Because, you know, after all, we're- we're humans, I guess it's a little bit selfish but, um, we're in charge right now. Um, and- and a lot of this- this view and a lot of the success stories in AI are really different from the things that you expect, you know, this, uh, this humanoid robot to come into your house and be able to do. For example this is a project from Stefano Ermon's group. Um, there's a lot of poverty in the world and, um, part of it is- is just kind of understanding what's- what's going on and they had this idea of using, uh, computer vision on satellite imagery to predict things like, you know p-, uh, GDP. Um, so this is obviously not a task that, you know, the- our ancestors in Africa were like, you know, getting really good at.

Um, but nonetheless it uses convolutional neural networks which is a technique that was inspired by, um, you know the brain and so that's- that's kind of interesting. Um, you can also have another application for saving energy by trying to figure out when to cool on datacenters. Um, as AI, is, uh, being deployed in more kind of mission critical s-, uh, situations such as self-driving cars or authentication. There are- there are a f- few th- new issues that come up. So for example, there are- thi- this phenomenon called adversarial examples, um, where you can take, um, these cool-looking glasses, you can put them on your face, and you can fool the computer, um, as- of- save our- our face recognition system to think that you're actually, you know, someone else. Um, or you can post these, uh, s- stickers on stop signs and you'd get this, uh- s- save our system to think that it's a, um, a speed limit sign. So there's obviously- there's- clearly these are, you know, big problems if we think about that the widespread deploy- deployment of AI. Um, there's also a less catastrophically but also p- pretty, um, you know, upsetting which is, uh, biases that you- many of you probably have read in the news about. So for example, if you take Malay which is a language that, uh, doesn't distinguish, um, in this writing form between he and she and you stick it into Google Translate. Um, you see that she works as a nurse but he works as a programmer, which is encoding certain, uh, societal biases, um, in the actual models. And one kind of an important point I wanna bring up is that, you know, it's it's how is machine learning and AI kinda working today? Well, it's, um, you know, society exists. Society is generating a lot of data. We're training on this data, and kind of trying to fit the data and try

and mimic what it's doing and then using predictions on it. What could possibly go wrong, right? Um, and so- so certainly people- a lot of people have been thinking about, um, how these biases are kind of creeping up and is an open and active area of research.

Something a little bit more, uh, kind of s- sensitive is, you know, asking well, these systems are being deployed to all these- all these people whether they kinda want it or- or want it or not. Um, and this, uh, this actually touches on, you know, people's, uh, you know, livelihoods. It actually impacts people's lives in a serious way. Um, so Northpointe was this company that developed a- a software called COMPAS that tries to predict how risky, um, criminal risk or how someone- how risky someone is essentially. Um, and ProPublica this organization realized whoa, whoa, whoa, whoa. You have this system that, uh, given an individual didn't reoffend is actually, um, more- twice as likely to classify blacks as incorrectly as, you know, non-blacks. So this is, uh, seems pretty problematic. And then Northpointe comes back and says actually, you know, I think we- I think we're being fair. Um, so given a risk score of 7, uh, we were fair because 60% of whites reoffended and 60% of blacks reoffended. Um, the- the point here is that there's- there's- there's actually no, um, solution to this in some sense sadly. Um, so people are finding or formulating different notions of fairness and equality between, um, how you predict or record it on different kind of, um, groups. But, um, or you can have different notions of fairness and which all seem reasonable from first principles but mathematically they can be, um, incompatible with each other. So this is- this is again an open area of research where we're trying to figure out as a society how, um, to deal with the schema that machine learning might be using these in kind of critical situations. Okay. So summary so far, um, there's an agent's view. Um, we're trying to really kind of dream and think about how do you get these capabilities like learning from very few examples that humans have into, you know, machines and a whole- maybe opening up a kind of a- a different set of technical capabilities.

But at the same time, and we really need to be thinking about how these AI systems are affecting the real world. And things like security, and biases, and fairness all kind of show up. It's also interesting to note that, you know, a lot of the challenges in deployment of an AI system don't really have necessarily to do with, um, you know, humans at all. I mean, humans are incredibly biased but that doesn't mean we want to build systems kind of in our- in, um, that mimic humans and kind of inherit all the kind of the flaws that humans have. Okay. Any questions about this? Maybe I'll pause for a moment. So let's go on. Um, so what I wanna do next is give an overview of the different topics, um, in the course. Um, and the way to think about all this is that, um, in AI we're trying to solve really complex problems. The real world is really complicated. And- but at the end of the day we want to produce some software or maybe some hardware that actually runs and does stuff, right? And so there's a very considerable gap between these things. And so how do you even approach something like self-driving cars or, um, you know, d- diagnosing diseases? You probably shouldn't just like go sit down at a terminal and start typing because then, um, there- there's no kind of- no overarching structure. So what this class is going to do is to give you one example of a structure which will hopefully help you approach hard problems, and think about how to solve them in a kind of more principled way. Um, so this is a paradigm that I call the, um, modeling inference and learning paradigm. Um, so the idea here is that

there's three pillars which I'll explain in a bit. And, uh, we can focus on each one of these things kind of in turn. So the first pillar is modeling. So what is modeling? The modeling is taking the real world, which is really complicated and building a model out of it. So what is a model? Model is a simplification that is mathematically precise so that you can, you know, do something with it, uh, on a computer.

Um, one of the things that's necessary is that modeling, um, necessarily has to simplify things and, you know, throw away information. Um, so one of the kind of, uh, the, you know, the art is to figure out what information to pay attention to and what information to keep. Um, so this is going to be important for example when you work on your final projects and you have a real world problem, you need to figure out, um, you can't have everything and you have to figure out judiciously how to, um, manage your- your resources. So here's an example. If you want to for example build a- a system that can find, uh, the best way to get from point A to point B in a graph- in a- in a city you can formulate the model as a- a graph where nodes are points in the city, and edges rep- represent ab- ability to go between these points with some sort of cost, um, on the edges. Okay. So now once you have your model you can do, uh, inference. And what inference means is asking questions about your model. So here's a model you can ask for example how- what is the shortest path from, um, this point, uh, to this point. Right. And that's because now your model land is a mathematically well-defined, uh, problem now you can- it's within the realm of, uh, you know, deve- developing algorithms to, you know, solve that problem. And most of the inference is ki- being able to do these computations, um, really efficiently. And finally learning addresses the problem, where does this model come from? So in any kind of realistic setting, um, the model might have a lot of parameters. Maybe it has, you know, millions of parameters and how do you s- if it- if it- wants to be faithful to the, you know, real world that how do you get all this, uh, information there. Um, manually p- encoding this information turns out not to be a good idea. This is, um, in some sense what, um, AI from the '80s was trying to do.

Um, so the learning paradigm is as follows. What we're gonna do is specify a model without parameters. Think about it as a skeleton. So in this case we have a graph but we don't know what the edge weights are. Um, and now we have some data. So maybe we have data of the form people tried to go from X to Y and they took 10 minutes, or an hour, or so on, um, and then from this data we can learn to fit the parameters of the model. We can assign, um, costs to the edges that kind of are representative of what the data is telling us, okay? So now in this way, we can write down a model without parameters, feed the data, apply a generic learning algorithm and get a model with parameters. And now we can go back and do, um, inference and ask questions, you know, about this. Okay. So this is kind of the- the- the paradigm. And I want to really emphasize that, you know, learning is not- as I've presented is really not about any one particular algorithm like nearest neighbors or neural networks. It's really a kind of a philosophy of how you go about approaching problems by defining a model and then not having to specify all the details but filling them in later. Okay. So here is the plan for the course. We're gonna go from low-level intelligence to high-level intelligence; and this is the intelligence of, um, of the, of the models that we're gonna be talking about. So first we're gonna talk about machine learning, and like I've kind of alluded

to earlier, machine learning is going to be such a kind of an important building block of- that can be applied to any of the models that we kind of develop. So the central tenet in machine learning is you have data and you go to model, its main driver of a lot of su- successes in AI because it allows you to, in software engineering terms, move the complexity from code to data. Rather than having, you know, a million lines of code which is unmanageable, you have a lot of data which is collected in kind of a more natural way and a smaller amount of code that can operate on this data and this paradigm has really been, it's really been powerful.

One thing to think about in terms of machine learning is that it, it is, requires a leap of faith, right. So you can go through the mechanics of down- downloading some machine learning code and you train them all but fundamentally it's about generalization, right. You have your data, you fit a model, uh, but you don't care about how it performs on that data; you care about how it performs on new experiences. And that leap of faith is something that's, um, I think gives machine learning its power but it's also a little bit, um, at first glance perhaps magical. Um, it turns out you can actually formalize a lot of this using, um, probability theory and, and statistics but that's kind of a topic for another time. Okay. So after we talk about machine learning, we're going to go back and talk about the, the simplest of models, right. So a reflex model is this. So here's a quiz. Okay. What is this animal? Okay, zebra. How did you get it so fast? Well, it's kind of a reflex, where your human visual system is so good, um, at, at doing these things without thinking. Um, and so reflex models are these, um, are models which just require a fixed set of computations. So examples like are linear classifiers, deep neural networks, um, and most of these models are the ones that people in machine learning um, use. Models is almost synonymous with, um, reflex on- in machine learning. The important thing that there's no feed for it. It just like you get your input bam, bam, bam, and here's your output. Okay, so that's, that's great because it's fast. But there's some problems that require a little bit more than that. Right. So for example here's another problem. Okay, quick, white to move. Where does she go? Okay, there's, there's probably like a few of you who are like chess geniuses, um, but for the rest of us, um, I have no idea. I don't know, wait, who's moving again? Um, so, so in these kind of situations, we need something perhaps a little bit more powerful than a reflex.

We need agents that can kind of plan and think, um, ahead. So the idea behind state-based models is that we model the world as a set of states which capture any given situation like, uh, a position in a, in a game and actions that take us between states which correspond to things that, um, you can do in the, in this game. Um, so a lot of game applications fall in this as category of robotics, motion planning, navigation. Um, also some things that are might not be- you might think of, um, planning as such as gen- you know, generation, um. In natural language or generating an image, um, you are, uh, can be cast in this way as well. So there's three types of state-based models each of which we'll cover in, um, you know weeks of time. So search problems are the classic, uh, you control everything so you're just trying to fi- find the optimal path. There are cases where there's randomness. For example if you're trying to go from point A to point B, maybe there's traffic that you don't, you know, don't know about or, um, in a game there might be dice that are- die which are rolled, and, uh, there's a third category which are adversarial games which is cases where your playing

an opponent who's actively trying to destroy you. So what are you gonna do about it? Um, so one of the games that we're gonna, uh, be talking about, uh, when we talk about games is Pac-Man; and one of the assignments is, um, actually building, um, a Pac-Man agent such as this. So, uh, while you're looking at this, think about how- what are the states and what are the actions and how would you go about you know devising a strategy for Pac-Man to eat all the dots and avoid all the ghosts? So that's something, uh, to maybe look forward to. There's also gonna be a competition. So we'll see how- who ends up at the top. Okay, so state-based models, um, are very powerful and a value to kind of have foresight.

Um, but some problems are not really most naturally cast as state-based models. For example, you know, how many of you play Sudoku or have played it before? So as the goal of Sudoku is to fill in these, uh, um, blanks with numbers so that, um, every row and column and three-by-three sub-block has the digits 1 through 9. So there's a bunch of constraints. Um, and there's no kind of sense in which you have to do it in a certain order, right. Whereas the, the order in how you move in chess or something is, you know, pretty important. Um, so, so these type of problems, uh, are captured by these variable-based models where you kind of think about a solution to the problem as an assignment to the individual variables, under some constraints. So constraint satisfaction problems, we'll spent a week on that, um, these are hard constraints. For example two people can be- or a person can't be in the two places at once for example. Uh, there's also Bayesian networks which we'll talk about which are variable-based models with, uh, soft dependencies. For example if you're trying to track, um, you know, a car over time, these are the positions of the car. These variables represent the position of the cars and these, uh, E's represent, uh, the- the sensor readings of the position of the car at that particular position and inference looks like trying to figure out where the car was given all this kind of noisy sensor reading. So that's also gonna be another assignment where you're going to deal with. Okay. So finally, um, now we get to high-level. What's- so what is high-level intelligence here? Um, and I put logic here, um, for a reason that you'll see clear. Yeah, is there a question? The Sudoku, can you explain why it's not a state-based model? Yeah, so the question is why is not the- why is the Sudoku problem not a state-based model? Um, you can actually formulate this as a state-based model, um, by just thinking about the sequence of, uh, assignments. But it turns out that, um, you can formulate in a kind of more natural way as a variable-based model which allows you to, uh, take advantage of some kind of more efficient algorithm to solve it.

Right, it's- think about these models as kind of different, um, analogy as like a programming language. So yes, you could write everything in you know C++ but sometimes writing in you know, Python or, or SQL for some things might be more- might be easier. Yeah. state based problem where you have both adversarial elements and an element of randomness? Yeah, so the question is how do you categorize state-based models where there is both randomness and an adversary? Um, we're also gonna talk about those as well. Um, and those would be- I, I would classify them as adversarial but there is also a random component that you have to deal with, games like backgammon. Yeah, question. Yeah, so the question is about whether, uh, some of these are more continuous and some of them are more discrete. Uh, I don't necessarily think of, uh, so a lot of the reflex models actually can work in continuous state spaces, for example images. Um, actually it's, it's almost a little

bit of the opposite where, um, the logic-based models are in some sense more, you know, discrete but you can also have continuous elements, you know, in there, um, as well. Um, so in this class, we're mostly going to focus on kind of discrete objects because they're just going to be simpler to work with. Okay, so what is this logic? So the motivation here is that suppose you, um, wanted a little companion who, um, you could boss around and, um, help or help you do things, let's say; that's a better way to say it. Um, so you'd like to be able to say okay, you know, tell us some information, um, and then later you wanna be able to ask some questions and have the system be able to reply to you. Um, so, um, you know how- how would you go about doing this? One way you could think about is building a system that you can actually talk to using natural language, okay.

So I'm actually going to show you a, a little demo, um, which, uh, is going to come up in the last assignment on logic; um, and well, let's see what you think about it. Uh, okay, so this is going to be a system that is, um, based on logic that I'm going to, um, tell the system a bunch of things and I'm going to ask some questions. So, um, I want you all to follow along and you see if you can, you know, play the role of the agents. Okay. So I'm going to teach you a few things like, um, Alice is a student, okay. So it says I learned something. Now let's, let's quiz, um, is Alice a student? Okay. Good. So that worked. Um, is Bob a student? What should the answer be? I don't know who's Bob. Um, okay. So now let's do, um, students are, uh, people. Um, Alice is not a person. I don't buy that okay. So, um, okay it's, you know, it's doing some reasoning, right? It's using logic, it's not, uh, just, um. Okay. So now, let's do, um, Alice is from Phoenix. Phoenix is a hot city. I know because I've lived there. Um, cities are places, and if it is snowing, uh, it is, um, then it is cold. Okay, got it. So, um, is it snowing? I don't know. Um, so how about this? Okay. So if, um, a person is from a hot place and it is cold, then she is not happy, okay. True. Right, um. I guess those of you who have spent all your live in California would maybe appreciate this. But, um, okay, so ho- is it snowing now? How many of you say yeah, it's snowing? How many say no? You don't know? Okay. Ah, ah, um, how about if I say Alice is, ah, happy. Okay, so is it snowing now? No, it should be no. Okay. So you, you guys were able to do this. Okay. So this is kind of an example of a interaction which, um, if you think about it has is ve- very different from where you would see kind of in a typical, um, you know, ML system where you have to show it millions of examples of one particular thing and it can do a kind of one task. This is much more of a very open-ended set of, um, I wish to say that the, the experiences are super rich but they're definitely diverse.

I teach- I just give one statement. I say it once and then all of a sudden it has all the ramifications and kind of consequences that built in and it kind of understands in a kind of a deeper level. Of course this is based on, you know, logic systems. Um, so it is brittle but this is kind of just a proof of concept to give you a taste of what I mean when I say logic. So, ah, these systems need to be able to digest this heterogeneous information and reason deeply with that information. And we'll see kind of how, um, logic systems can do that. Okay. So that completes the tour of the topics of this class. Um, now I want to spend a little bit of time on course logistics. Uh, so I wanna- all the details here are online. So I'm not going to be complete in my coverage, um, but I just wanna give you a general sense of what's going on here. Okay. So what are we trying to do in this course? Um, so prerequisites, um, there's

programming, um, discrete math and, ah, probability. So you need t be able to code and you need to be able to, um, do some math and, uh, some kind of basic proofs. Right? So these are the classes that are, um, required or at least recommended that you- or if you have some equivalent experience that's, you know, fine too. Um, and what we- what should you hope to get out of this course? Right. So one had- the course is meant to be giving you a set of tools using the modeling inference learning paradigm. It gives you a set of tools and a way of thinking about problems that hopefully will be really useful for you when you go out in the world and try to solve real world problems. Um, and also by- as a side product I also want all of you to be more proficient at your math and programming because those are kind of the core elements that, ah, enable you to do kind of interesting, you know, things in AI. So a lot of AI and you, you read about it, it's very flashy but really the foundations are still, um, just you know math and programming in some sense.

Okay. So the coursework is homeworks, exam, and a project. That's what you have to do, um, Homeworks, there's eight homeworks. Each homework is a mix of writing- written and programming problems centered on a particular application covering one particular type of model essentially. Um, like I mentioned before there's a competition for extra credit. There's also some extra credit problems in the, in the homeworks, um, and when you submit code, we're gonna run- we have an auto-grader that runs. It's gonna run on all the test cases but you get a feedback of only a subset. So you can, um, it's like, you know, in machine learning, you have a train set, and you have a test set. So don't train on your test set. Okay. So um, the exam is, ah, testing your ability to use the knowledge that you learn to solve new problems. Right. So there's, um, I think it's worth taking a look at exam because this, this kind of surprises people every- the exam is a little bit different than the types of problems that you see on, on the homework and there are kind of more problem, you know, solving. So the exam isn't going to be like a multiple choice like, okay, you know, um, you know, when was Perceptrons published or something like that. It's gonna be, here's a real life problem. How do you model it and how do you come up with a solution? Um, they're all going to be written. It's closed book except for you have a one page of notes and this is a great opportunity to actually, um, review all the material and actually learn the ah, the content in the class. Um, so the project I think is a, a really good opportunity to take all the things that we've been talking about in the class and, um, try to find something you really care about and try to apply it. Work in groups of three and I really recommend finding a group early, um, and as I emphasize it's your responsibility to find, you know, a good group. Right? Um, don't come to us later like one week before the project deadline and say, "Oh, you know, my group members they, um, they ditched me," or something.

We really try to, try to nail this down use Piazza to- or your other social networks to find a good group. So throughout the quarter there's going to be these milestones for the projects. So, um, to prevent you guys from procrastinating into the very end, um, so there's gonna be a proposal where you try and brainstorm some ideas, progress report, a poster session which is actually a whole week before the final report is due, um, and the project is very open. So this can be, um, really liberating but also might be a little bit daunting. Um, we will hopefully give you a lot of structure in terms of saying okay, how do you define your task? How do you implement different, um, baselines or oracles? Which I'll explain later. How do

you evaluate? How do you, um, analyze what you've done? And each of you will- each project group will be assigned a CA mentor, ah, to help you, ah, through the process and you're always welcome to come to my office hours or Dorsa's, or any of the CAs to get additional, um, help either brainstorming or figuring out what the next step is. Ah, some policies, ah, all assignments will be submitted on Gradescope, um, there are seven total late days you can use, and most two per assignment. After that there's no credit. Um, ah, we're gonna use Piazza for all communication so don't email us directly. Leave a post on Piazza. If- I encourage you to make it public if it's, it's not sensitive, but if it's, you know, personal, then obviously make it private, um, and try to help each other. We'll actually award some extra credit for students who help answer, um, other student's questions. So all of the details are on the course website. Okay. So one last thing and it's really important and that's the Honor Code. Okay. So especially if you're, um, you know, you've probably heard this if you've been at Stanford.

If you haven't, then I wanna really kind of make this clear. So I encourage you all to have- collaborate, discuss together. But when you- when it comes to actually the homeworks, you have to write up your homework and code it independently. So you shouldn't be looking at someone's writeup. You shouldn't be looking at their code. Um, and you definitely shouldn't be copying code off of GitHub. Um, um, that's hopefully should be, you know, obvious and maybe less obvious, you should not- please do not post your homework assignments on GitHub. I know you're probably proud of the fact that your Pac-Man agent is doing really well but please don't post on GitHub because then that's going to be our Honor Code violation. Um, when debugging, um, with- if you're working together, it's fine to as long as it's kind of looking at input-output behavior so you can say to your partner, "Hey, I put in this, um, input to my test case and I'm getting a 3. What are you getting?" So that's fine but you can't. Remember don't look at each other's code. Um, and to enforce this, we're gonna be running MOSS, which is a software program that looks for code duplication, um, to, to make sure that, ah, the rules are being followed and, you know, changing one variable name is- or you'll be so- anyway enough said. Just don't, don't, don't do that. Okay? Any questions about this? I wanna make sure this is important or about any of those logistics. Yeah. The final project, ah, you can put on GitHub. Yeah. Yeah. Yeah, private GitHub repos, uh, is fine. Yeah, question in the back? Is it necessary to have a group or can you do a solo project? Uh, the question is can you, can you do a solo project? You can do a solo project, you can do a project with two people, or you can do a project with three. I would encourage you to try to work in, uh, groups of three because you'll be able to do more as a group, and there is definitely, uh, you know, it, it, it's not like if you do a solo project we'll be expecting like one third of the, the work.

So okay. Anything else? All right. Okay. So in the fi- final section, I want to actually delve into s- some technical details. Um, and one thing we're going to focus on right now is, um, the, kind of inference and learning components of, of this course. So I'm going to talk about how you can approach these through the lens of, you know, optimization. So this is going to be, uh, it might be a review for some of you but hopefully, it's gonna be a, a good, um, you know, way to get everyone on the same page. Okay. So what is optimization? There's two flavors of optimization that we care about. There's, uh, Discrete Optimization, where you're

trying to find the best, uh, discrete object. For example, you're trying to find the best, uh, path or the path P that minimizes the cost of that path. Um, we're going to talk about one algorithmic tool, um, based on Dynamic Programming which is a very powerful way of solving these, um, complex optimization problems. Um, and the key, you know, property here is that the set of paths is huge and you can't just, uh, trial them and compute the cost and choose the best one. So you gonna have to choose something clever. The second brand of optimization is continuous optimization and formally this is just finding the best of vector of real numbers that satisfies or minimizes some objective function. So a typical place this shows up is in learning where you define, uh, objective function like the training error and you're trying to find a weight vector W. So this notation just means it's a list of numbers, D numbers that minimizes the training error. And we're going to show that gradient descent is, uh, uh, easy and a surprisingly effective way of solving these, um, continuous optimization problems. Okay. So to introduce these two ideas, I'm going to look at two, um, problems and trying to kind of work through them. So this might be also a good, um, you know, way to think about how you might go approach a, you know, homework problems.

And I'll try to kind of talk you through this, um, in a bit more detail. Okay, so the first problem is, um, you know, computing edit distance. Um, and this might not look, you know, like an AI problem, but a lot of, ah, AI problems have this as kind of a, you know, building block if you wanted to do some sort of matching between, um, you know, two words or two, um, biological sequences. So the input is you're given two strings. Um, we're gonna start writing over here on the board just to work this out. So given two strings, um, S and T. Um, so for example, um, a cat and um, the cats. Okay. So these are two strings and you wanna find the minimum number of edits that is needed to take transform S into T. And by edits I mean you can insert, um, a character like you can insert S, you can delete characters, I can delete this A and you can substitute one character for another. So you can replace this A with a T. Okay. Um, so here's some examples. What's the edit distance of cat and cat? It's 0, you don't have to do anything. Cat and dog is 3, cat and at is 1, you insert the A or insert a C. Um, cat and cat is 1, um, and a cat and the cats is 4. Okay. So the challenge here is that there are, ah, quite a different number of ways to insert and delete. Right, so if you have a string of- that's very long there's just way too many things to like just try out all of them. Okay, so then, how do we, how do we go about, um, coming up with a solution? So any ideas? Yeah. simplify the output in terms of saying that the substitution tells us we considered deletion peoples who considered a substitution or vice-versa by saying like an empty character. Yeah, yeah. So let's try to simplify the, the, the problem a bit. And building up on your what you, um, what was said. So, um, one thing to note is that okay, where so the general principle, let me just write the general principle, um, is to, you know, reduce the problem to a simpler problem because then you can hopefully solve- it is easier to solve, and then you can maybe keep on doing that until you get something that's trivial.

Okay. So there's maybe two observations we can make. One is that well, we're technically saying we can, um, you know, insert into S right but if we insert into S, it makes the problem kind of larger in some sense, right? I mean that's not, that's not good. That's not reducing the problem. But, but whenever we insert into S, um, we probably want to insert things which are in T. We wanna like cancel something out, right? So we wouldn't insert a K there

for any reason. We probably wanna insert a S in which case no S matches that and then we've reduced that problem, right? So we can actually think about, you know, inserting into S to S as equivalent to kind of deleting from, um, from T. Okay, does that make sense? All right. So another observation we can make is that, you know, we can start inserting anywhere. We can start inserting here and then jump over here and to this. But this just introduces a lot of, um, you know, ways of doing it which all kind of result in the same answer. So why don't we just start more systematically at one end and then just proceed and try to chisel-off the problem, um, kind of let's say from the end. Okay, so start at the end? Okay, so, so now we have this problem and to draw a problem in a little box here. Um, so let's start at the end. Yeah, question. What's the reasoning used to reach that principle start at the end? . the question is why are we starting at the end as oppo- well, the idea is that if you start at the end then you have kind of a more systematic and consistent way of, you know, reducing the problem. So you don't have to think about all the permutations of where I can delete and substitute. Why is it more systematic to go from the right to the left than from the left to the right? We can also do it left to right. So the end or the start is both fine. This is just- I just picked the end. Yeah. Are we not starting at the end and then give us the optimal strategy? Yeah, the question is how do we know that starting, um, at one end can give you the optimal strategy? Um, so, you know, if you wanted to prove this more rigorously there's some work but, um, I'll just try to give you a, you know, an intuitive answer.

Um, suppose you didn't start at the end, and you just made a sequence of steps like I insert here, I delete here, and then I went over here and um, did all those operations to S. I could have equivalently also just sorted those by, you know, where it was happening and then just proceeded from one end to the other, and I would arrive at the exact same answer. So without loss of generality, I can start at that. Any other questions? Okay. So yeah. Instead of doing this wouldn't the more viable approach be that trying to recognize some patterns instead of doing this. I think between the two strings "s" and "t" like some form of- some sort of pattern string. Yeah. So the question is, maybe you can recognize some patterns. Uh, it's like okay, oh, cat. That's- that's- maybe those should be lined up. Um, I guess these examples are chosen so that these patterns exist, but we want to solve the problem for cases where, um, the pattern might not be obvious. So it could be- we want to work it for- it to work for all strings. Maybe there is no pattern, and we still would want to- kind of an efficient algorithm to do it. Yeah. Can't we just like use dynamic programming? Like we go one by one, there was always like - Yeah. Either we're doing, um, substitution, or, um, otherwise it's like the same character. Or we have to insert- Yeah. - um, and then we keep going, and you just like remember each like to- to strings that we have at one point- Uh-huh. -so that if we calculated that we don't have to do it again. Yeah. Yeah. That's it. Yeah. Yeah. Yeah. Great idea. Let's do dynamic programming. Um, so that's what I'm kind of trying to build up from- uh, build up to. Okay so, um, so if you look at this- so dynamic programming is a kind of a general technique that essentially allows you to express this more complicated problem in terms of a simpler problem.

Uh, so let's start with this problem. If we start at the end, um, if the two match then, well we can just immediately, um, you know, delete these two and that's- it's gonna be the

same, right? So we can get- we are gonna get some free rides there. Okay, but when they differ, um, now we have many options. So what we could- what could we do? Well, we could, um, um, you know substitute. Okay, we can change the "t" to an "s". So what does that leave us with? So I can do a cat, "t" is the- the cat, the- Okay, so I can substitute. Um, okay. Um, what else can I do? Someone say something I can do. So I can insert, um, insert where into- So I can insert an "s", right? Yes. But that's the same as, you know, deleting from "t". So by, uh- you can basically also just delete this "s". Um, so this is our cat, and I deleted this "s" from "t". Okay, so this is, um, let's call it, uh, you know, um, I guess let's call this insertion- it's technically insertion . And then finally what can I do? I can also remove "t". So a, ca, the, cats. Okay, so this is delete. And right now you're probably looking at this like, well, obviously, you know, you sho- you should do this one. But in general it's hard to tell. What if I just give you some arbitrary strings, you know, who knows what the right answer is. Um, so in general how do you pick? Yeah. In the second one, the "t" is supposed to be for cats. You mean this one? Yeah. So here I inserted an "s", right? But then because there's two s's here, I just canceled them out and what was left So you can think about this as really deleting from- What if I'm considering Like in the original problem you said we're transferring "s" to "t". Yeah. Yeah. Yeah. So, um, um, because of this I'm kind of trying to re-frame the problem a little bit. Okay, so which one should I choose? Yeah.

What about the substitution the other way? Um, the substitution the other way meaning change- "s" to "t". Sorry there's too many s's and t's here which is going to be a bit unfortunate. And then replace the last s in cats with "t". Oh, you could- How do we eliminate that Um, that's- you can think about that as kind of equivalent. So, if you identify two letters that you want to make the same, then you could- you can replace the one to be the other, or the other to be that. I mean if- officially we've been kind of framing it as we're only editing "s" which is the reason that it's asymmetric. Okay, so which one of these? Door "a" door "b" or door "c"? Yeah. Would you look between "s" and "t" for every step because there's "cat" in both of them? Yeah, so you could try to look inside but, um, but remember these are- might be really complicated. So you- we wanna kind of a simple mechanized procedure to tell. What about the next letter? The next letter. "t" Um, yeah let's- let's pretend these are- you- you can't see inside them. Okay. . Keep going with each of the different cases. Yeah, okay, so let's keep on going. So, I'm not going to draw everything, but you can also try to break this down into- maybe there's three actions here, and three actions here. All right. Um, and at the end of the day you hopefully have a problem that's simple enough, that, um, where "s" equals "t' or something then you're done. Um, but then, you know, how- how do I- how do I know? Suppose I've solved this. Suppose if someone just told you, okay, I know this cost, I know this cost, I know this cost. What- what should you do? Yeah, you should take the minimum, right? Like remember we want to minimize the edit distance. So, um, there's three things you can do. Each of them has some costs of doing that action which is, you know, one. Every edit is the same cost. And then there's a cost of, you know, continuing to do whatever you're doing. And so we're just gonna take the minimum over those.

Yeah. How do we know that that's, like- that's the maximum amount of distance that we have to take? Yeah, so I was trying to argue that, um, with- if you're going to right to left,

it's, uh, without loss of generality. Because if you've- went left to right, or in some other order, you can also replay the edits, um, in order. one letter that you needed one assertion like like upstream. But if you went from like the left it looks like as if you're . Yeah. Okay. Yeah. I think it works. Um, okay, so- so let's, um, try to code this up and see if we can make this program work. Okay, so, um, I'm gonna do editDistance. Can everyone see this? Okay, so, um, so I'm gonna define a function that takes two strings, and then I'm going to um, define a recurrence. So, recurrences are- are, I guess, one word I haven't really used, but this is really the way you should th- kind of think about, uh, dynamic programs, and this idea of taking complex problems and breaking it down. It's gonna show up, in you know, search problems, MDPs, and, you know, games. So, I guess it's something that you should really be comfortable with. So, let's um, define recurrence, uh, as follows. Um, so remember at any point in time, I have, uh, let's say a sub problem, and since I'm going right to left, I'm only considering the first, um, "m" letters of "s" and the first letter "n" letters of "t". Okay, so recurse is going to return the minimum edit distance between two things, the first "m" letters of "s", and the first "n" letters of "t". Um, I'm gonna post this online so you guys don't have to, like, copy- try to copy this. Um, okay, so, um, okay, suppose I'm gonna- I'm gonna define this function. Uh, if I have this function what should I return? Recurse of-. So "m" is an integer, right? So "n" is an integer, so I'm going to return the length of "m" and the length of "n". Okay, so that's kind of, uh, the initial state. Sorry. Yup. Okay. Um, All right. So now you need to fill out this function.

Okay, so let's- let's um, consider a bunch of cases. So here's some easy cases. Suppose that, um, "m" is zero, right? So I have- comparing an empty string with something that has "n" letters. So, what should the cost of that be? I heard some mumbling. . It should be "n" and symmetrically if "n" is 0 then result should be "m", um, and then if now we come to the kind of initial case that we consider which is the end match a match. So, if "s" um, the last letter of "m", you know, this is 0-based indexing. Um, so that's why there's a minus 1. So, this matches. Then what should I do? So, now we reduce this to a sub problem, right? So, I have "m" minus 1 and "n" minus 1. Okay. And now comes the fun case which we looked at. So there's- um, in this case the last letter doesn't match. So, I'm gonna to have to do some sort of edit, can't just let it slide. Yeah. Question. Would you- do you need a full "s" to "t" compare or "s" through "m" and then "t" through "n" to compare? Worse than doing a full s, a compare. rather than waiting until, um, first- Yeah. -stream at the last slide than that. There- there's probably a way you can make this more efficient. I'm just gonna try to get the basic thing in there. Okay. So substitution. Okay. So what's a cost of a substitution? I pay 1 to do the substitution, but and in- as a reward I get to, um, reduce the problem to n minus 1 and n minus 1, right? So I lop off a letter from s and I lop off a letter from t. So what else can I do? So I can, um, you know, delete. So that also costs 1. And when I delete, I delete from s and then n. So this remains the same. And then now you can think about the insertion, um, is n minus 1, right? Because remember insertion into s is deletion from t, that's why this is n minus 1. Okay. And then the result is just gonna be a minimum of, uh, all these things. Okay. Return result. Okay. So just, uh, and then, how do I call this function? Um, a cat, the cats. So let me print out the answer. Um, let's see if it works.

Okay. Print out 4. Therefore, I conclude it works now. I mean if you were doing this, uh, you would probably want to test it some more, but in the interest of the time, I'll kind of move on. So let me just kinda refresh. Okay. So I'm computing this at a distance between two strings and we're gonna define a recurrence that works on sub problems, where the sub problem is the first m letters of s and the first n letters of t. And the reason I'm using integers instead of, um, strings is to avoid like string copying, um, implementation detail, but it doesn't really matter. Um, so base cases. So you wanna reduce your problem to a case where it's- it's trivial to solve. Um, and then we have the last letter matches. And then we have a letter doesn't match and you have to pay some sort of cost. I don't know which action to take. So I'm gonna take them, you know, minimum of all of them. And then I call it by just calling, you know, recurse. Okay. So this is great, right? So now I have a working thing. Um, let's try another test case. So I'm gonna make this. Um, so if I do times 10, this, uh, basically, uh, replicates this string 10 times. So it's a- it's a long string-longer string. Okay. So now I'm gonna run it. Maybe I shouldn't wait for this. Is there a base case? Um, there is a base case, I- I think that it expanded- it's- what- what's wrong with this code? Very slow. Um, yes, it's very slow. Why is it slow? Yeah, right? So- so I'm recursing. Every point recurses three times. So you kind of get this exponential, you know, blob. Um, so there's kind of a- how do you solve this problem? Yeah. You can memo I think I heard the word memoize, which is another way to kind of think about. Memorize plus, um, I guess, recurrences is dynamic programming, I guess. Um, so I'm gonna show you kind of this, um, way to do it which is pretty, uh, uninvasive. Um, and generally I recommend people. Well, get the slow version working and then try to make it faster. Don't try to be, you know, too slick at once.

Okay. So I'm gonna make this cache, right? And I'm gonna say if m, n is in the cache, then I'm gonna return whatever's in the cache. So cache is just a dictionary mapping. Um, the key which is, um, identification of the problem I'm interested in solving, and the result which is the answer that I computed. So if I already computed it, I don't need a computer again, just return it. And then at the end, if I have to compute it, then, um, I have to put this in the cache. Okay? So three lines or four lines, I guess. Yeah. Yeah. That's a great point. Uh, this should be outside of the recurse object. Yeah. Glad you guys are paying attention. Um, otherwise, yeah, it would do basically nothing. Any other mistakes? Yeah. Um, there is also function decorators that like implement memoizing for you. In this class, are you okay if we use that or would you rather us like make our own in this case? Um, you can use the deco- you can be fancy if you want. Okay. Um, yeah. But- but I think this is, you know, pretty transparent. Easy for learning purposes. Okay. So let's run this. So now it runs instantaneously as opposed to- I actually don't know how long it would have taken otherwise. Okay. And sanity check for t is probably the right answer because there's four was the original answer and multiply by 10. Okay. any other questions about this? So this is an example of, you know, kind of basic, uh, dynamic programming which are, uh, you'd solve a problem trying to formulate it as a recurrence of a complicated problem in terms of smaller problems. Um, and like I said before this is gonna kind of show up, um, um, over and over again in this class. Yeah. Yeah. So the question is why does this reduce, uh, redundancy. Is that right? Um, so maybe I can do it kinda pictorially. Um, if you think about, let's say, you have a, um, a problem here, right? And this gets, um, you know, reduced to, um, um, I'm

just making kind of a arbitrary, um, diagram here. So this problem gets reduced to these two.

And this problem gets reduced to these two, um, and- and so on, um, right? So if you think about- if you didn't have memoization, you will just be paying for the number of paths. Every path is a kind of you have to compute from scratch. Whereas, if you do memoization, you pay the number of nodes here, which a lot of this has shared like here. Um, you know, once you compute this, no matter if you're coming from here or here, you're kind of using the same value. Okay. So let's- let's move on. So the second problem, um, we're gonna talk about is, uh, has to do with continuous optimization. And the motivating question here is how do you do, um, regression? Which is a kind of a bread and butter of, um, you know, machine learning here. So here we go. Regression. Okay. So imagine you get some points. Okay, so I give you a point which is 2, 4. Then I give you another point, let's say 4, 2. And so these are data points, you want to, let's say, predict housing price from, you know, square footage or something like that. You want to predict health score from, um, your blood pressure and some other things. So this is pretty common in machine learning. And the question is how do you fit a line? I'm going to consider the case where your line has to go through the origin, just for simplicity. Um, so you might want to like find, you know, a fit. Two points is maybe kind of a little bit degenerate, but that's the simple example we are going to work with. In general you have lots of points and you want this to fit the line that best kind of, uh, is close to the points. Okay, so how do you do this? So there's a principle called least squares, which says, well, if you give me a line which is given in this case by a slope w, I'm going to tell you how bad this is. And badness is measured by looking at all the training points, and looking at these distances. Right. So here I have, you know, this particular, uh, a particular, let's say point x_i. If I hit it with a w, then I get, basically the, uh, you know, the y-intercept here, not the y-intercept but the- like the y value here.

That's my prediction. The real value was y_i, which is, you know, up here. And so if I look at the difference, I want that difference to be zero. Right. So in, in least squares, I square this, and I say, I want this to be as small as possible, right. Now, this is only for one point. So I'm going to look at all the points. Let's suppose I have n points, and that's a function that I'm going to call f of w, which basically says, for a given weight vector, which is a slope, give me a number that characterizes how bad of a fit, um, this is. Where 0 means that I fit everything perfectly, and large numbers mean that I fit poorly. Okay? All right. So, so that's your regression. So how do I solve a regression problem? So how do I optimize this? Can you do this in your head? So if I actually had these two points, what should w be? Okay, it doesn't matter. We'll, we'll compute it. So how do we go about doing this? So one principle, which is maybe another general takeaway is, abstract away the details. Right. Um, this is also true with the dynamic programming, but sometimes, you know, you get- if you're too close to the board, and you're looking at, oh man, these, these points are here and I need to fit this line. How do I do that? You kind of get kind of a little bit stuck. Why don't we think about this f, as say some function? I don't, I don't really care what it is. And let's plot this function. Okay. So now this is a different plot. Now, this is, ah, the weight, and this is f of w. Always label your axes. And let's say this function looks like this. Okay. So which means that for this slope, I pay this, you know, amount, for this slope, I pay this amount and, and so on. And

what I want to do, I want to minimize f of w, which means, I want to find, um, the w which, um, has the least value of f of w, right? Question? Okay. So you take the derivative.

So what is the derivative giving you? It tells you where to move, right? So if you look over here, so you can- in general, you might not be able to get there directly, in this actually particular case you can because you can solve it in closed form, but I'm going to try to be more general. Um, so you start here. This, this derivative tells you, well, the function is decreasing if you move to the right. So then you should move to the right. Whereas over here, if you end up over here, the derivative says, the function is decreasing as we move to the left. So you should move to the left, right? So what I'm going to introduce is this, uh, algorithm called gradient descent. It's a very simple algorithm. It basically says, start with some place, and then compute the derivative, and just follow your nose. Right? If the derivative says it's negative, then just go this way. And now you're on a new point, you compute the derivative again, you descend, and now you compute it again. And then maybe you compute the derivative and it says keep on going this way and maybe you overshoot, and then you come back. And then, you know, hopefully you'll end up at the minimum. Okay. So let's try to see what this looks like in code. So gradient descent is one of the simplest algorithms, but it really underlies essentially all the algorithms that you people use in machine learning. So let's do points. We have two points here. Um, and I'm going to define, um, some functions. Okay, so f of w, so what is this function? So I'm going to sum over all the different, um, you know, and basically at this point it's converting math into Python. So I'm going to look at all the points. So for every x, y, what the model predicts is w times x minus y. And if I square that, that's going to be the error that I get on that point. Then, if I sum over all these errors then I get my objective function. Okay. Array of- so yeah. So you can put array here if you want, but it doesn't matter. It's, it's actually fine.

Okay. So now I need to compute the derivative. So how do you compute the derivative? So if your calculus is a little bit rusty, you might want to brush up on it. So what's the derivative? Re- remember we're taking the derivative with respect to w, right? There's a lot of symbols here. Always remember what you're taking derivative with respect to. Okay. The derivative of the sum is the sum of the derivative. So now I need to take the derivative of this. Right. And what's the derivative of this? Something squared, um, you bring the two down here, and now you multiply by the derivative of this. And what's the derivative of this? Should be x. Right? Because this is a- y, this is a constant, and w derivative- w times x with respect to w is x. Okay. So that's it. Okay, so now let's do gradient descent. Let's initialize with w equal 0. Then I'm going to just, um, you know, iterate a hundred times. Normally, you would set some sort of stopping condition, but let's just keep it simple for now. Okay, so for every moment, I'm going to- I have a w, I can compute the value of the function, and also take the gradient of the derivative. Gradient just means derivative in higher di- dimensions, which we'll want later. Um, okay. And then, what do I do? I take, uh, w, and I subtract the, the gradient. Okay. So remember- okay, I'll be out of here. Okay. So, uh, I take the gradient. Remember I want to have the gradient. Uh, gradient tells me where the function is increasing, so I want to move in the opposite direction. And eta is just going to be this, uh, step size to, um, keeping things under control. We'll talk more about that next time. Okay, so now, I want to do, print out what's going on here. So iteration, print out the

function, and t value. Okay. All right, so let's compute the gradient. And, um, so you can see that the iteration, we first start out with w equal 0. Then it moves to 0.3, then it moves to 0.79999999 and then it looks like it's converging into 0.8. And meanwhile, the function value is going down from 20 to 7.

2 which happens to be the optimal answer. So the correct answer here is 0.8.

References

Bodie, Z. (1976), “Common stocks as a hedge against inflation”, The Journal of Finance,Vol. 31, No. 2, pp. 459- 470.

Brunie, C. H., Hamburger, M. J., & Kochin, L. A. (1972), “Money and stock prices: the channels of influence”, The journal of Finance, Vol. 27, No. 2, pp. 231-249.

Büttner, D., Hayo, B. and Neuenkirch, M., 2011. The impact of foreign macroeconomic news on financial markets in the Czech Republic, Hungary, and Poland. Empirica, 39(1), pp.19-44.

Chen, N. F., Roll, R., & Ross, S. A. (1986), Economic forces and the stock market, Journal of business, Vol. 59, No. 3, pp. 383-403.

Ciner, C. (2001), “On the long run relationship between gold and silver prices A note”, Global Finance Journal, Vol. 12, No. 2, pp. 299-303.

Coˆte´, D. et al., 2004. The performance and robustness of simplemonetary policy rules in models of the Canadian economy. Canadian Journal of Economics, p. 978–998.

Economic Impact of Covid-19 on Different Sectors of Indian Economy. PRAGATI : Journal of Indian Economy 7, no. 2 (2020). doi:10.17492/jpi.pragati.v7i2.722021.

Enders, W. (2004) Applied Econometric Time Series, 2nd Edition. In: Wiley Series in Probability and Statistics, John Wiley & Sons, Inc., Hoboken.

Fama, E.F. 1981. Stock returns,real activity, inflation and money. American Economic Review.

Granger, C. W. J.. (1969). Investigating Causal Relations by Econometric Models and Cross-spectral Methods. Econometrica, 37(3), 424. .

Johansen, S. (1988). Statistical Analysis of Co-integrating Vectors. Journal of Economic Dynamics and Control, 12, 231-254.

Johansen, S. (1988). Statistical Analysis of Co-integrating Vectors. Journal of Economic Dynamics and Control.

Mishkin, F. S., 2007. Housing and the Monetary Transmission Mechanism.

Molodtsova, T. & Papell, D. H., 2008. Out-of-Sample Exchange Rate Predictability with Taylor Rule Fundamentals. Journal of International Economics, pp. 167-180.

Ortner, S. B. Resistance and the problem of ethnographic refusal. Comparative studies in society and history, 37(01), 1995.13.

Pearce, D.K. and Roley, V.V. 1988. Firm characteristics, unanticipated inflation, and stock returns. Journal of Finance, pp. 965-981.

Pethe, A., and Karnik, A., (2000), Do Indian Stock Markets Matter? Stock Market Indices and Macro -Economic Variables, Economic and Political Weekly, 35(5), p. 349.

Rossi, B., 2013. Exchange Rate Predictability. Journal of Economic Literarture, pp. 1063-1119.

Saikkonen, P., & Lütkepohl, H. (2000) Testing for the Cointegrating Rank of a VAR Process with Structural Shifts. Journal of Business & Economic Statistics;18(4):451. doi:10.2307/1392226.

SATTI, A. U. H. & MALIK, W. S., 2017. The Unreliability of Output-Gap Estimates in Real Time. The Pakistan Development Review, pp. 193-219.

Seeborg M C, Jin Z, Zhu Y. The new rural-urban labor mobility in China: Causes and implications, The Journal of Socio-Economics, 2000.

Sprinkel, B. W. (1964), “Money and Stock Prices”. Richard D. Irwin, Homewood III.

Visually (2012) Effective Internet Marketing - what is working for Marketers”, 2012, (http://visual.ly/effectiveinternet-marketing, (20.11.2015).