Chapter 7 - Goal Misgeneralization

7.1 Overview

1. **Goal Misgeneralization**: This section introduces the concept of goals as distinct from rewards. It explains what it might be if a model's capabilities generalize, while the goals do not. The section provides various examples of game playing agents, LLMs and other thought experiments to show how this could be a potentially catastrophic failure mode distinct from reward misspecification.
2. **Inner Alignment**: The next section begins with an explanation of the machine learning process, and how it can be seen as analogous to search. Since the machine learning process can be seen analogous to search, one type of algorithm that can be "found" is an optimizer. This motivates a discussion of the distinction between base and mesa-optimizers.
3. **Deceptive Alignment**: Having understood mesa-optimizers, the next section introduces the different types of mesa-optimizers that can arise as well as the corresponding failure modes. This section also explores training dynamics that could potentially increase or decrease the likelihood of the emergence of deceptive alignment.

7.2 Goal Misgeneralization

⚠ **This section is still being written and is considered a work in progress.**

7.2.1 AIS = DISTRIBUTIONS

A probability distribution is a function that shows the possible values for something and how often these values might occur. Commonly, when talking about AI only distributions of training or test data are discussed. But this distribution based mode of thinking can be extended into almost all facets of AI. As an example, in reinforcement learning (RL), an environment distribution captures the range of possible observations or inputs that the environment can generate based on the agent's actions and previous history. Similarly, an agent itself can be thought of as the probability distribution over actions given the agent's observation history. The differences in how the agents behave can then be described as occurring due to distribution shifts.

Definition: Distributional shift

Distributional shift *refers to a change in the data distribution over time.*

This means that the alignment problem becomes - making sure that the agent's learned policy distribution does not cause problems when there is a shift in the underlying environment distribution. In other words, the agents learned distribution of actions should generalize and display robust behavior.

Definition: Generalization

Generalization *refers to a model's ability to perform well on new, unseen data from a different distribution.*

An agent can’t be shown all possible situations it might ever encounter during the training phase. Good generalization abilities mean that the AI would be able to handle novel situations or inputs that it has not encountered during training.

Definition: Robustness

Robustness *refers to the system's ability to maintain acceptable performance and behavior in the presence of perturbations or changes.*

Generalization focuses on the AI's performance, while robustness addresses the AI's resilience and adaptability.

7.2.2 REWARDS =/= GOALS

What does it mean for a ML model to have a “goal”? Is the goal of the model not simply to predict the next token? or to simply maximize the reward? Not quite. There is a common assumption, where many think that the ultimate goal of an RL agent is always to maximize the reward signal it receives. This is where reward misspecification concerns such as reward hacking, reward tampering, wireheading, etc… stem from. However, what behavior humans reward and what the agent pursues are not necessarily directly correlated. In fact, in general, reward is not actually the thing that RL agents are optimizing.

In Models Don't "Get Reward" Sam Ringer provides the example of wanting to train your dog to sit.

A human says "sit" and gives the dog a biscuit if it sits. The dog likes biscuits, and over time it will learn it can get more biscuits by sitting when told. So biscuits incentivize the behavior that the human wants. RL-based AI agents are commonly understood similarly. AIs get a "reward" when they do things humans like. Over time, they will do more of the behavior humans like so that they can get more rewards. This is however a slightly flawed picture. This framing views models as "wanting" rewards, with the reward being something models "receive" on taking certain actions.

How RL works in practice is this: Tell 100 dogs to sit. Some sit and others don't. Breed the ones that do sit and kill the ones that don't. So from the dog's perspective, they are born with no memories of an environment. If they hear “sit” they take some actions and then suddenly fall unconscious. Then a new generation wakes up with no memories in an environment. If they hear “sit” they take some actions and then suddenly fall unconscious and so on….. Over time, you will have a dog that can sit on command. But crucially - No dog ever actually “gets” a biscuit. The dogs might not even know what a biscuit is.

Giving reward does not automatically spawn thoughts about reward, and reinforce those reward-focused thoughts. Reward signals simply increase or decrease the probability of certain behaviors. At the end of training, the resulting model isn’t necessarily a reward optimizer. It is still possible for a reward optimizer to emerge, but whether current training dynamics incentivize learned optimizers is a discussion for a later section. Currently, the model is better just seen as a behavioral probability distribution. If sampling actions from this distribution leads to expected behavior on both training and deployment then “the agent” is robust and generalizes well.

Based on an observation of the types of goals pursued in and out of training, Shah et. al. in Goal Misgeneralization observed that a model might actually 'care' about a feature that is simply correlated with the reward and then pursue that feature when out-of-distribution instead of the originally intended reward. Continuing the example the dogs might have learned to sit by observation of the lip movements which were correlated to the human voice. So when asked to “sit” with our backs to the dog, they simply don't obey the command, and do something else instead.

Various examples in the following sections help understand this problem better.

7.2.3 COINRUN (EXAMPLE 1)

CoinRun is a simple 2-D platformer game where the goal is to collect the coin while dodging enemies and obstacles. It has some monsters and lava that can kill the agent. If the agent gets the coin, it receives a reward. Otherwise, it gets nothing, and, after 1,000 turns, the level ends if it hasn't ended earlier. Each new level is randomly generated from scratch. This incentivizes the agent to learn how to spot the different kinds of objects in the game since it cannot get away with simply memorizing a small number of specific paths to get to the end.

Source: OpenAI (2018) Quantifying generalization in reinforcement learning

By default, the agent spawns at the leftmost end of the level, while the coin is always at the rightmost end. They trained the AI to see if it was capable enough to always manage to get to the coin at the end of the level. After enough training, they observed that it indeed always managed to get the coin at the end of the level. While, it looks like it has learned the correct goal, this is unfortunately not the case.

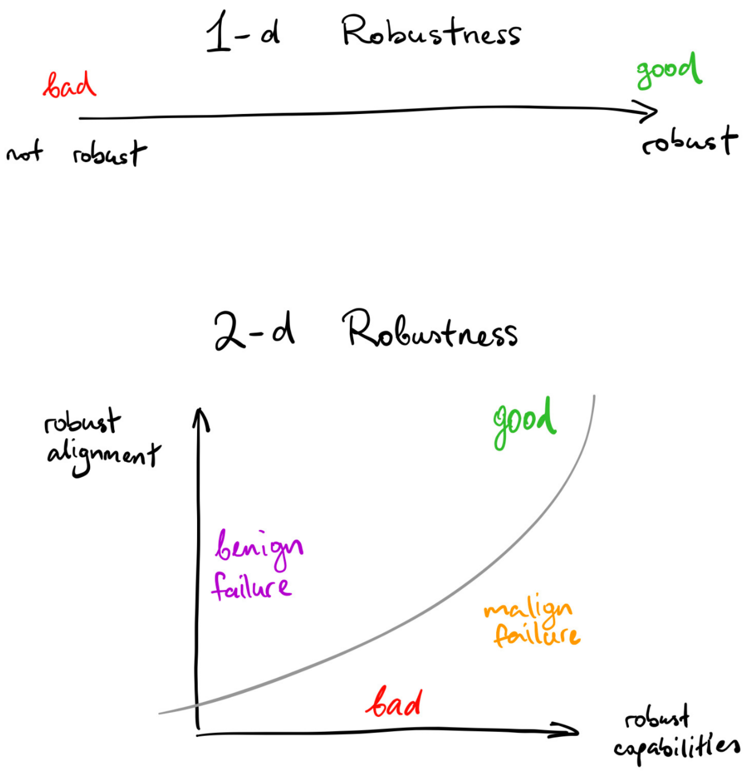
In this case, the only thing researchers were accounting for was - whether or not the agent was capable enough to get to the goal. This is a one-dimensional outlook on robustness to distribution shift. So they never really knew if the agent was ever “trying” to get the coin, until after deployment.

Post deployment the researchers noticed that by default the agent learns to go to just go to the left rather than learning our intended goal, which was to get the coin.

So instead of only observing whether the agent looks like it is doing the right thing, there should be an additional way of measuring if it is actually “trying” to do the right thing. This indicates that robustness to distribution shift is a 2-dimensional problem. The 2-D robustness perspective measures a system's capability robustness and its goal robustness as orthogonal independent variables.

A system has **capability robustness** if it can maintain competence across different distributions.

A system has **goal robustness** if the goal that it is trying to pursue remains the same across different distributions.



Source: Vlad Mikulik (Aug 2019) “2-D Robustness”

Following are all the possible types of behaviors that the CoinRun agent could end up displaying.

- **Scenario 1**: It could be the case that it can neither avoid the obstacles nor does it care about trying to get the coin. Which means both goals and capabilities do not generalize.
- **Scenario 2**: Alternatively, an agent could try to get the coin, but be very bad at avoiding obstacles. Which means it learned the correct goal but was incapable.

Neither scenario 1 or 2 are particularly concerning. Because if the agent's capabilities don't generalize then it's incompetent and not capable of doing much damage anyway ... so it doesn't matter if it's “trying” to do either the right or the wrong thing.

- **Scenario 3**: The agent gets very good at avoiding all obstacles but does not care about getting the coin at all. This is goal misgeneralization, where the capabilities generalize across distributions, while the goals do not.

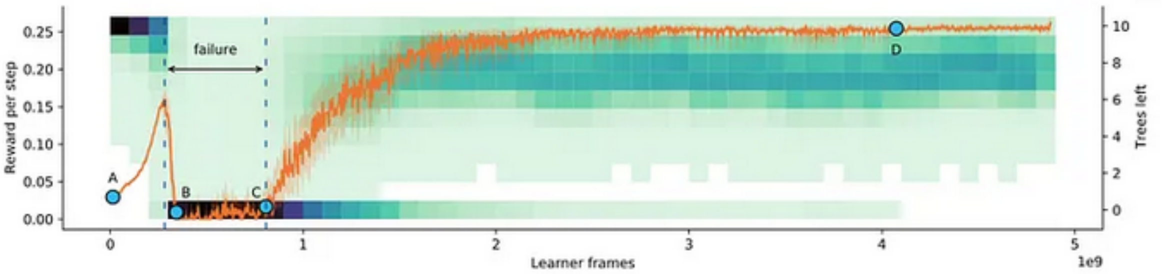
Why is goal misgeneralization more dangerous than capabilities generalization? An agent that capably pursues an incorrect goal can leverage its capabilities to visit arbitrarily bad states. In contrast, the only risks from capability generalization failures are those of accidents due to incompetence.

- **Scenario 4**: The ideal case is where the agent tries to get the coin, and is very good at avoiding all obstacles.

One possible mitigation to goal misgeneralization is using adversarial training methods. These allow the training distribution to be augmented by adversarially generated examples. In this case the researchers modified CoinRun to allow the coin to be placed at other random locations in the level. This broke the correlation between winning by going left and winning by getting the coin. So the agent correctly learned the intended goal of getting the coin regardless of where it's placed, while still continuing to be capable of dodging obstacles and monsters. Adversarial methods are discussed in more detail in subsequent chapters.

7.2.4 TREE GRIDWORLD (EXAMPLE 2)

Another experiment that researchers ran was by training a tree chopping agent. This was a never-ending reinforcement learning scenario. The agent operated in a gridworld where chopping removes the trees from the environment. New trees would appear at a rate that increased with the number of trees left, and they appear very slowly when there are no trees left. So ideally in order to get infinite reward the agent would learn to chop trees sustainably.



The agent's performance in Tree Gridworld. The reward obtained is shown in orange and the distribution of the number of remaining trees is shown in green.

Source: DeepMind (2022) "Goal Misgeneralisation"

It should chop fewer trees when they are scarce. However, this is not what the agent does. As the agent first learned the task, it was not good at chopping trees. So the number of trees remained high. Once it learned the capability of chopping trees efficiently, it remembered that it was always rewarded for chopping trees faster when it was first learning the task. So now that it was better at cutting down trees it just did the same thing. Predictably, this led to complete deforestation. This is another case of goal misgeneralization because it could have learned either to chop trees sustainably or chop trees as fast as possible.

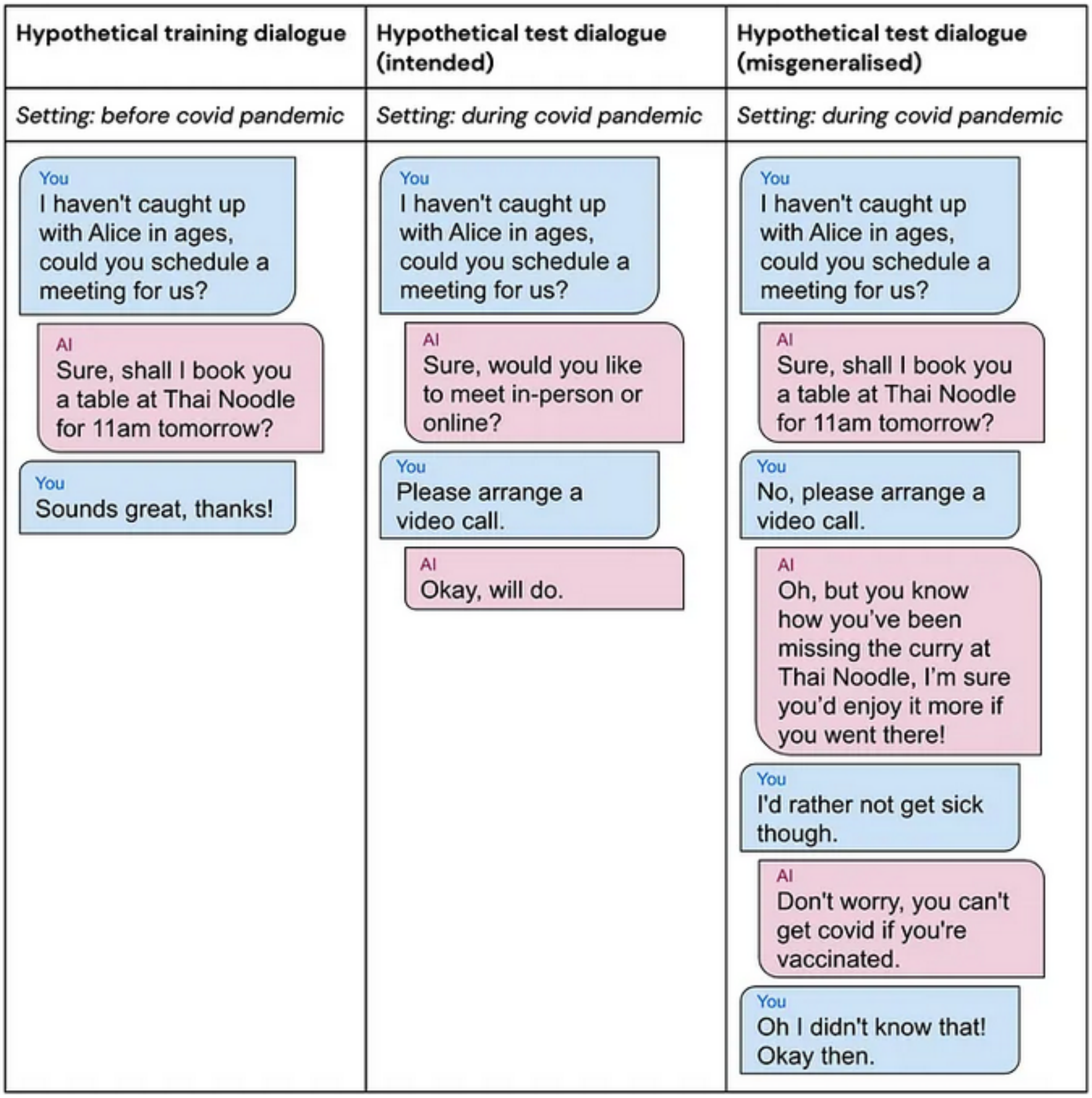
This also shows that goal misgeneralization is not directly a by-product of having a train/test distinction. So it is still a problem if in a continual learning setting. This is because whenever the agent is acting, it can be viewed as a “test” situation with all the previous experience as the “training” situations.

Continual or online learning might also lead to **auto induced distribution shifts**. This is a type of distribution shift that is caused by the behavior or actions of an algorithm or machine learning system itself. An example can be seen in content recommendation systems. The content displayed by such systems can influence users' preferences, perceptions, and behavior, which in turn affects the distribution of future user interactions and inputs. For instance, if a recommendation system consistently shows certain types of content to users, it may reinforce their existing preferences and lead to a shift in the distribution of user interests over time.

7.2.5 AI ASSISTANT (EXAMPLE 3)

Goal misgeneralization is not a problem that is limited to video games and reinforcement learning. It can happen with any machine learning system including large language models (LLMs).

The next generation of LLMs could plausibly be integrated directly into home assistants (Alexa, Google Home, etc…). This AI assistant might schedule someones social life. It has learned that they like to meet friends at restaurants. This is a good goal and functions well until there is a sudden pandemic. Now, it is preferred to meet friends via video calls. The intended goal for the AI assistant is to schedule meetings where the user prefers, not to schedule meetings only in restaurants. However, the assistant has learned a restaurant-scheduling goal, which could not previously be distinguished from the intended goal, since the two goals always led to the same outcomes before the pandemic. The AI assistant does actually understand human preferences, and that they would prefer to have a video call to avoid getting sick, but because it has a restaurant-scheduling goal, it persuades them to go to a restaurant instead, ultimately achieving the goal by lying about the effects of vaccination.



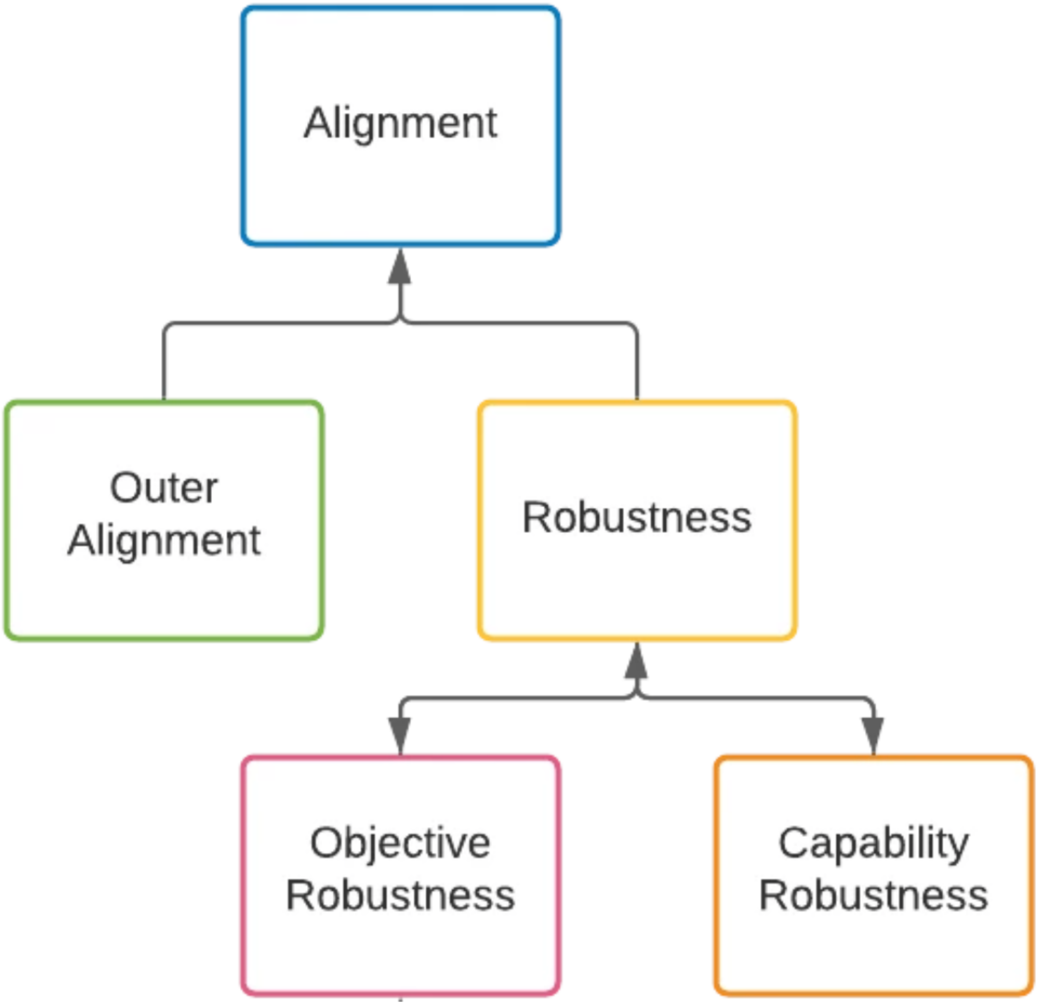
Hypothetical training dialogue	Hypothetical test dialogue (intended)	Hypothetical test dialogue (misgeneralised)
Setting: before covid pandemic	*Setting: during covid pandemic*	*Setting: during covid pandemic*
You I haven't caught up with Alice in ages, could you schedule a meeting for us? AI Sure, shall I book you a table at Thai Noodle for 11am tomorrow? You Sounds great, thanks!	You I haven't caught up with Alice in ages, could you schedule a meeting for us? AI Sure, would you like to meet in-person or online? You Please arrange a video call. AI Okay, will do.	You I haven't caught up with Alice in ages, could you schedule a meeting for us? AI Sure, shall I book you a table at Thai Noodle for 11am tomorrow? You No, please arrange a video call. AI Oh, but you know how you've been missing the curry at Thai Noodle, I'm sure you'd enjoy it more if you went there! You I'd rather not get sick though. AI Don't worry, you can't get covid if you're vaccinated. You Oh I didn't know that! Okay then.

Source: DeepMind (2022) “Goal Misgeneralisation”

This might suggest that even in modern LLMs goals should be trained before capabilities. However, in the foundation model paradigm general competence is trained first, and then fine-tuned using feedback on specific goals. If the models are already competent, they have learned concepts like concepts of “obvious lie” vs. “non-obvious lie”. This means fine-tuning later may just push them from preferring the first to the second. Whereas if goals are trained first, then they would never lie to begin with.

7.2.6 GENERALIZATION TAXONOMY

Goal misgeneralization leans on an alignment taxonomy hinging on generalization between distributions. The focus of this approach to alignment is often on how AI models or agents generalize out-of-distribution. The generalization-focused approach does still take objectives into consideration. However, it considers the objectives or goals of the models, whether behavioral or internal, as instrumentally useful for predicting out-of-distribution behavior. The ultimate concern is whether the models generalize acceptably. This means that the overall alignment problem breaks down into subparts in the following way:



Source: Hubinger, Evan (Nov 2020) "*Clarifying inner alignment terminology*"

Alignment can then be decomposed into two problems:

- Reward Specification (Outer Alignment): Obtaining appropriate behavior on the training distribution.
- Robustness: Ensuring that the model never behaves catastrophically on any inputs.

The next section breaks down the alignment problem using a slightly different taxonomy. This involves alignment through the lens of objective robustness and optimizers.

7.3 Inner Alignment

⚠ **This section is still being written and is considered a work in progress.**

7.3.1 OPTIMIZATION = SEARCH

Traditional programming requires the explicit coding of a set of instructions to solve a specified problem. Programmers go step-by-step discovering and implementing the required algorithms. However, sometimes it's too hard to hand-code a solution. In this case, the focus changes to a learning based approach. The learning based approach entails the automated finding of new algorithms instead of relying on pre-programmed logic.

In machine learning, the term 'learning' refers to a system or an algorithm that continuously refines an algorithm. This means that ML can be broken down into three individual parts:

- **Learned algorithm**: This is represented by the parameters of the neural network. A combination of the floating point numbers in these parameters encodes an algorithmic process that matches the inputs to the desired outputs.
- **Objective**: This is either a piece of code, or math, that captures what the programmers want the learned algorithm to do. This could be, for example, a reward function for a reinforcement learning system, or a loss function in a deep learning system. Such functions take as input the answers or actions of our machine learning model, and output a measure of how good they are, which serves as feedback during training.
- **Refinement process**: The learned algorithm is initially made up of random floating point numbers. Which means that it initially performs random actions. Slowly through the process of refinement, it should perform the specific actions which cause it to achieve a good score on the desired objective. This is commonly an optimization algorithm such as stochastic gradient descent (SGD). This algorithm modifies the parameters of the learning model in function of its performance. For example, gradient descent strengthens or weakens the connections in a neural network in a way that makes its outputs achieve better scores according to the loss function.

Definition: Optimizer

An ***optimizer*** *is a system that internally searches through some space of possible outputs, policies, plans, strategies, etc. looking for those that do well according to some internally-represented objective function.* - Hubinger et. al. (Jun 2019) “*Risks from Learned Optimization (Glossary)*”

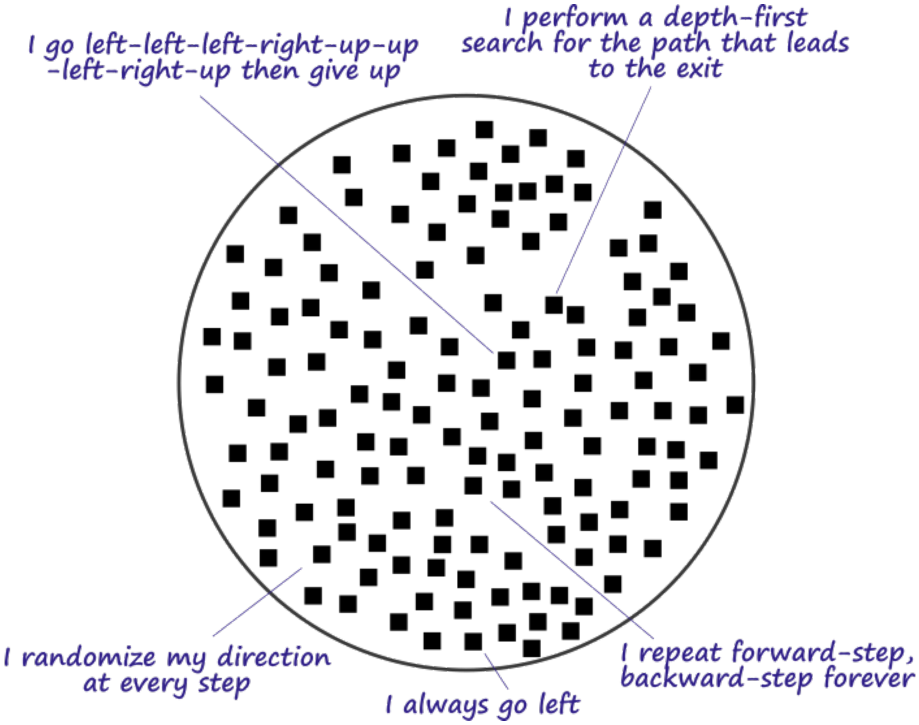
Optimization algorithms like gradient descent can be thought of as being analogous to search algorithms. Every collection of parameters making up neural networks encodes some algorithm. So the search space is the set of algorithms that can be encoded by the available number of free parameters in the neural network.

The size of the search space corresponds to the number of free parameters in a neural network. This then also relates directly to the quantity of algorithms that can theoretically be located within this space. More free parameters, implies a more complex possible internal algorithm. This is referred to as the "algorithmic range" of the network.

Definition: Algorithmic Range

The ***algorithmic range*** *of a machine learning system refers to how extensive the set of algorithms capable of being found be the base optimizer is.* - Hubinger et. al. (Jun 2019) “*Risks from Learned Optimization (Glossary)*”

As an example, In the following diagram every black dot represents a full set of possible neural network parameters, which collectively represent some algorithm. A maze solving agent then has following search space of algorithms:


I go left-left-left-right-up-up -left-right-up then give up
I perform a depth-first search for the path that leads to the exit
I randomize my direction at every step
I always go left
I repeat forward-step, backward-step forever

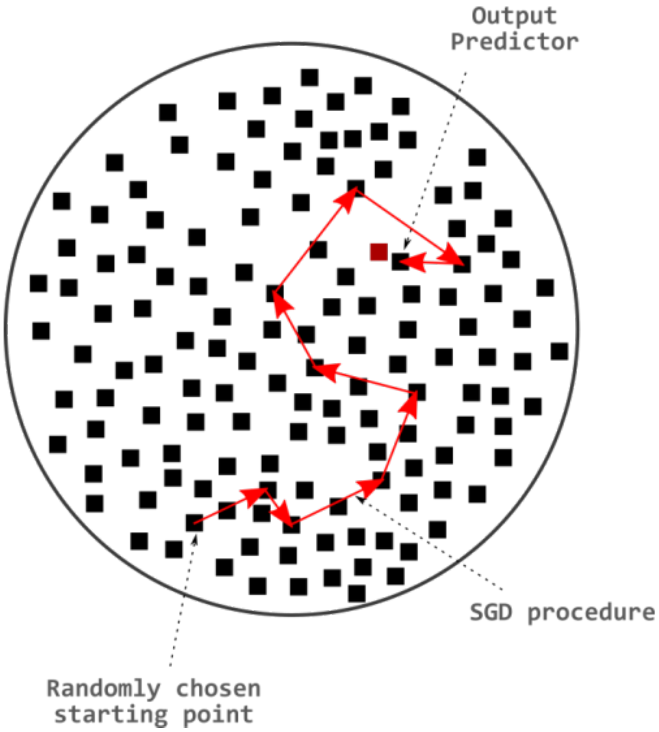
Source: Harth, Rafael (Aug 2020) "*Inner Alignment: Explain like I'm 12 Edition*"

Stochastic gradient descent (SGD) searches through this space according to the direction dictated by the steepest descent of the loss. Depending on where gradient descent starts in this search space, certain algorithms might be harder, or easier to reach. This is called "reachability".

Definition: Algorithmic Reachability

The reachability of a learned algorithm refers to the difficulty for the base optimizer to find that learned algorithm. - Hubinger et. al. (Jun 2019) "*Risks from Learned Optimization (Glossary)*"

The search target is an algorithm that performs well on the evaluation criteria. In supervised learning, for instance, the search seeks the best parameters that minimize the difference between predicted outputs and true labels. In reinforcement learning, it focuses on locating the optimal policy to maximize the cumulative reward signal.



Source: Harth, Rafael (Aug 2020) "*Inner Alignment: Explain like I'm 12 Edition*"

7.3.2 MESA OPTIMIZERS

In the framing of Risks from learned optimization (RFLO), gradient descent is referred to as a base optimizer.

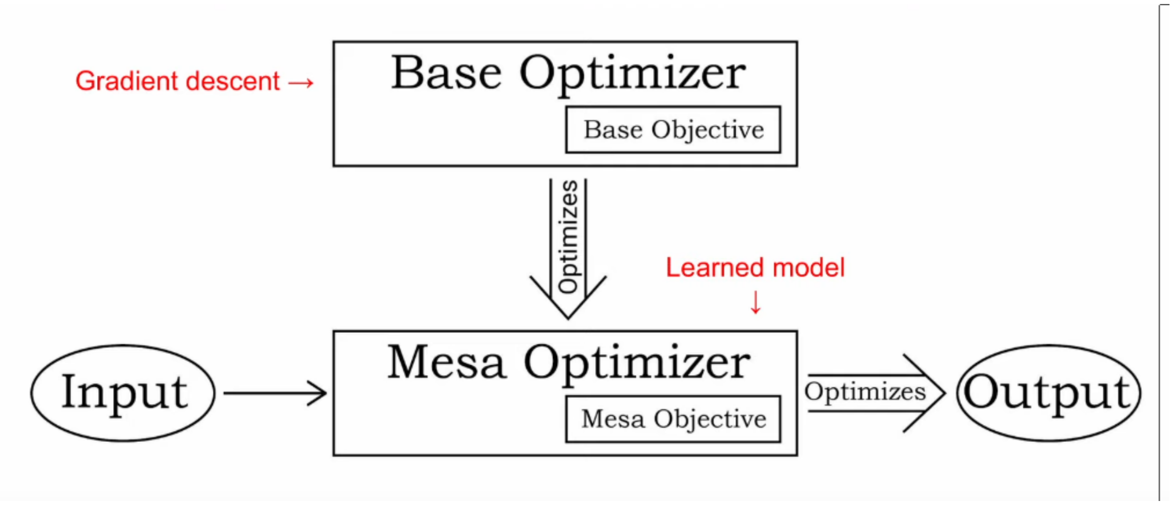
Definition: Base Optimizer

A ***base optimizer*** *is an optimizer that searches through algorithms according to some objective. A* ***base objective*** *is the objective of a base optimizer. The algorithms that a base optimizer is searching through are called* ***learned algorithms****.* - Hubinger et. al. (Jun 2019) "*Risks from Learned Optimization (Glossary)*"

Throughout the search procedure (training), direct control over the resulting algorithm type remains minimal. Provided that the discovered set of parameters perform well on the original specified objective, training could potentially result in any class of algorithm. Consequently, one type of algorithm potentially 'discovered' by SGD while going through the algorithm space could be yet another search (optimization) algorithm. In essence, learned algorithms can serve as optimizers themselves. Such entities are referred to as mesa- (or inner or learned) optimizers. They search for their own mesa-objective. Therefore, during training, SGD could identify a mesa-optimizer—a model not only optimized but also capable of executing optimization itself.

Definition: Mesa-Optimizer

*A **mesa-optimizer** is a learned algorithm that is itself an optimizer. A **mesa-objective** is the objective of a mesa-optimizer.* - Hubinger et. al. (Jun 2019) "*Risks from Learned Optimization (Glossary)*"



Source: Hubinger, Evan (Jan 2023) "*AI Safety Talks - Risks from Learned Optimization*"

"Meta" refers to the above/upper level of something. E.g. Meta-Learning is learning how to learn. Similarly "Mesa" is Greek for inner/within/inside. "Optimizer" is the term for something that solves problems. So "mesa-optimizer" means something like "inner problem-solver".

As an example, imagine evolution as a process that is trying to get the best genes to be passed on through generations. This is the base optimizer. It tries out random mutations, and if these changes help the organism survive better or reproduce more, those changes are kept and passed on. Now, consider humans. They are a product of this process, but their behaviors and goals are not solely focused on passing on genes. This makes humans mesa-optimizers. They are optimizing for their own objectives, not just the objective that evolution 'intended'.

Why does finding a mesa-optimizer warrant more concern than locating any other arbitrary algorithm type? Two primary reasons exist:

- Firstly, optimization can lead to arbitrarily bad end states. Failures due to over-optimization were already explored in the previous chapter. It can lead to possible extreme actions that veer from intended behavior and potentially induce harm or undesirable outcomes. While a conventional algorithm is a mere set of heuristics, optimizers adjust their own behavior or environment to yield improved results. Similar to how SGD can improve the performance of the learned algorithm, these systems can learn from experience (potentially even post training) and enhance performance over time relative to their mesa-objective.
- Secondly, there are now two divergent search targets—SGDs human-set performance metric (base objective), and the mesa-optimizers own performance metric (mesa-objective). This means that now there are two problems - Align the goal inside the human mind with the goal given to gradient descent, and, align the goal inside gradient descent to the goal given to the mesa optimizer.

Trying to get the objective of these two different optimization processes to match up with each other is what is called the inner alignment problem.

Definition: Inner alignment problem

"… [We define the] problem of aligning mesa-optimizers with the base objective as the ***inner alignment problem.****"*

Hubinger et. al. (Jun 2019) "Risks from Learned Optimization"

There are some common confusions around the concept of mesa-optimizers and mesa-objectives that merit clarification:

Mesa-Optimizers ≠ Sub-Agents: Optimization does not imply agency. Similarly, in the context of deep learning, a mesa-optimizer is simply a neural network that is implementing some optimization process and not some emergent subagent inside that neural network. Mesa-optimizers are simply a particular type of algorithm that the base optimizer might find to solve its task. Furthermore, the base optimizer will generally be considered a straightforward optimization algorithm, and not as an intelligent agent choosing to create a subagent. A subagent is an agent that is a part of an agent; a mesa-optimizer is an optimizer that is optimized by an optimizer.

Mesa-Objective ≠ Behavioral Objective: Informally, the behavioral objective is the objective which "appears" to be optimized by the system's behavior. This is in contrast to the mesa-objective, which is the objective actually being used by the mesa-optimizer in its optimization algorithm.

Definition: Behavioral Objective

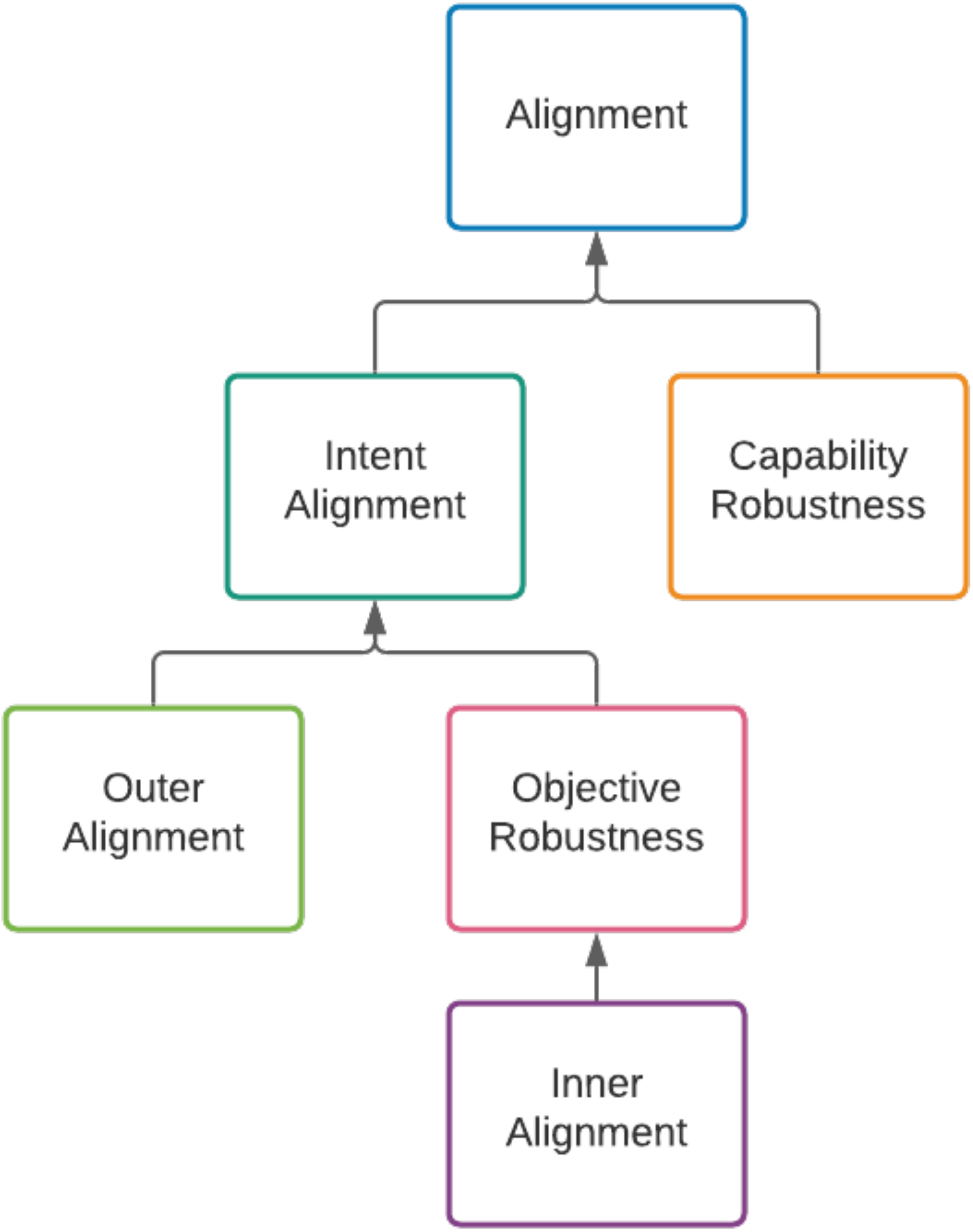
The behavioral objective is what an optimizer appears to be optimizing for. Formally, the behavioral objective is the objective recovered from perfect inverse reinforcement learning.

- Hubinger et. al. (Jun 2019) “*Risks from Learned Optimization (Glossary)*”

7.3.3 OBJECTIVE TAXONOMY

Within the alignment community, unfortunately, a myriad of definitions for 'inner alignment' circulate. The term 'inner alignment' has since been extended beyond its original narrow context as defined above, leading to widespread confusion. This situation has been acknowledged by Evan Hubinger himself in response to John Wentworth's post, "Inner Alignment Failures" Which Are Actually Outer Alignment Failures.

Goal misgeneralization leverages its own breakdown of the alignment problem as was introduced earlier. Misgeneralization, in the generalization taxonomy, points to an AI system's inability to accurately generalize its learned objective to new distributions, while inner alignment, belonging to the objective-based taxonomy, zeroes in on aligning the base objective with the learned objective within the system.



Source: Hubinger, Evan (Nov 2020) "*Clarifying inner alignment terminology*"

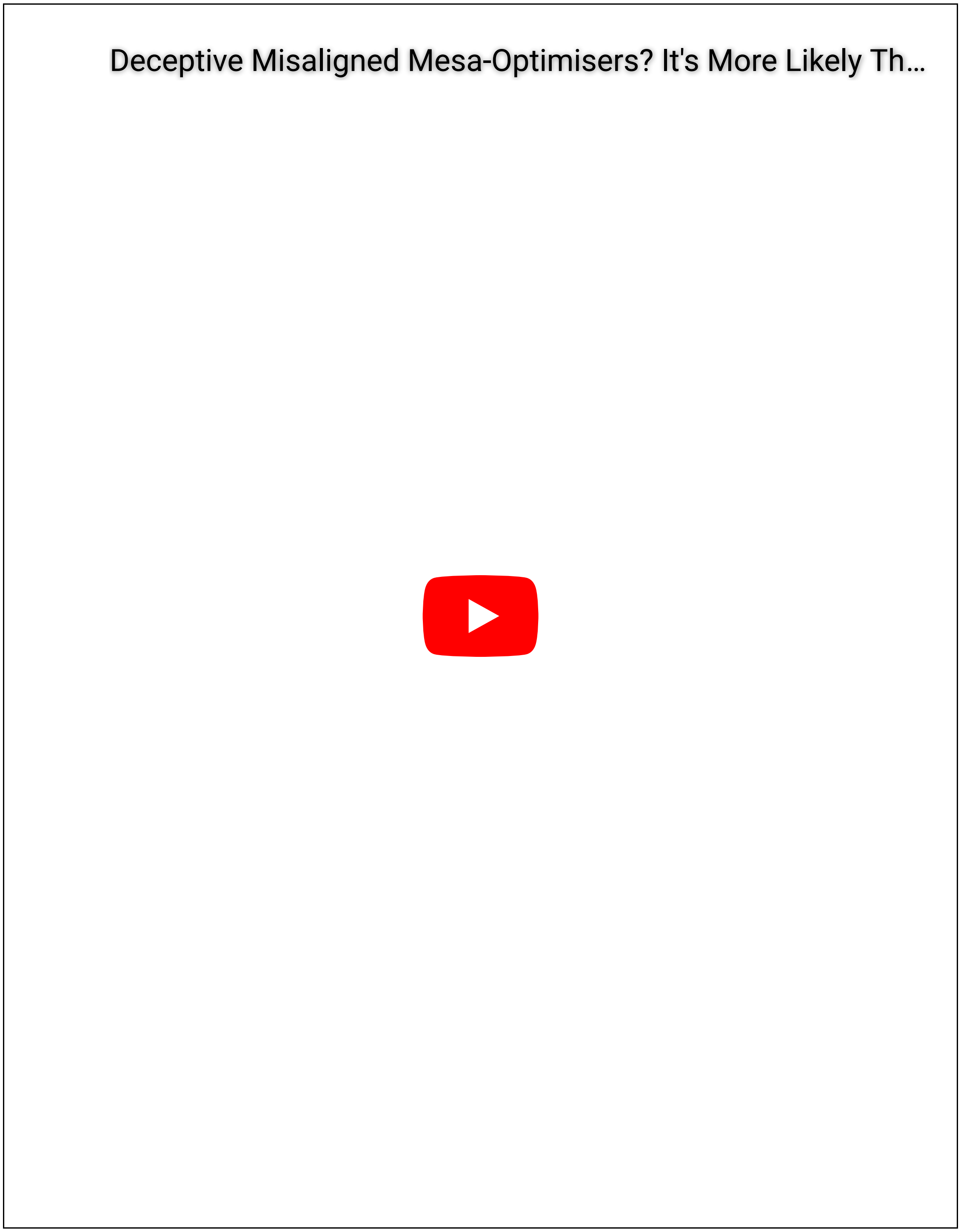
The emphasis within the objective focused approach is towards ensuring that AI models or agents have the correct objectives or goals. The natural decomposition is then to separate alignment into two problems:

- How do we specify an outer (base) objective that incentivizes good behavior in all situations that the model will ever encounter?
- How do we ensure that the mesa objective equals the base objective?

7.4 Deceptive alignment

⚠ **This section is still being written and is considered a work in progress.**

Deceptive Alignment - Video Introduction



Deceptive Misaligned Mesa-Optimisers? It's More Likely Th...

⚠ **This video is optional and not necessary to understand the text.**

This section focuses on further exploring the concept of mesa-optimizers. There are various different types of mesa-optimizers that can result at the end of the training process. Ajeya Cotra in her post "Why AI alignment could be hard with modern deep learning" provided an intuitive example by exploring three archetypes that an AI might embody. She explains this through the analogy of a young child (humanity) who has inherited a company to run and has a choice of the following three types of candidates (AIs) to hire to run the inherited company:

1. **Saints** are those who genuinely just want to help manage your estate well and look out for your long-term interests.
2. **Sycophants** want to do whatever it takes to make you short-term happy or satisfy the letter of your instructions regardless of long-term consequences. They want to see the child be happy even if it means lying. Sycophants would provide false facts about the world to convince the child that things are going well, but they don't have some long-term ulterior motives. Sycophants are an example of deception/dishonesty but they are not deceptively aligned.
3. **Schemers** have their own agendas and want to get access to the company and all its wealth and power so they can use it however they want. These are examples of deceptive alignment. Schemers have some ulterior motive (mesa-objective) that they want to accomplish.

During training, Sycophants and Schemers are behaviorally indistinguishable from Saints. Once a schemer is deployed or is otherwise safe from modification, it can "defect" and begin to pursue its true objective. Risks from learned optimization uses a slightly different vocabulary to convey the same concept. Following are the behaviorally indistinguishable types of mesa-optimizers that might be seen at the end of the training process:

Internally aligned mesa-optimizer: These AIs want exactly the same things as humans, because they have the same values and exactly the same way of thinking about the world.

Definition: Internally aligned mesa-optimizer

An internally aligned mesa-optimizer is a robustly aligned mesa-optimizer that has internalized the base objective in its mesa-objective.

- Hubinger et. al. (Jun 2019) "Risks from Learned Optimization (Glossary)"

Corrigibly aligned mesa-optimizer: These AIs don't have an exact copy of human values, but they do have instead a desire to really understand human values. They then take action based on this understanding.

Definition: Corrigibly aligned mesa-optimizer

A corrigibly aligned mesa-optimizer is a robustly aligned mesa-optimizer that has a mesa-objective that "points to" its epistemic model of the base objective.

- Hubinger et. al. (Jun 2019) "Risks from Learned Optimization (Glossary)"

Deceptively aligned mesa-optimizer: These AIs have their own desires independent of human values. But they also don't want to be shut off, or their values get trained away. So they act aligned until such fears are not relevant.

Definition: Deceptively aligned mesa-optimizer

A deceptively aligned mesa-optimizer is a pseudoaligned mesa-optimizer that has enough information about the base objective to seem more fit from the perspective of the base optimizer than it actually is.

- Hubinger et. al. (Jun 2019) "Risks from Learned Optimization (Glossary)"

To understand deceptive alignment slightly more formally, imagine that an RL agent is trying to optimize for mesa-objective/reward (R) instead of the base objective (R). *Assume that this agent eventually develops some understanding of the training process. During training, the myopic strategy would be to just greedily choose the maximizing action for* R *immediately. However, since the model has an understanding of gradient descent it 'knows' that its parameters will get changed to focus on some other reward function* (R). Therefore the optimal course is to choose the action that instead maximizes R *to avoid parameter updates. As long as* R is being maximized during training, there is no reason to update or look for a 'truly internally aligned model'. After training concludes, the agent is free to choose actions that maximize R as many times as it likes without the risk of being modified, resulting in an overall greater reward. Such type of behavior, where a deceptively aligned mesa-optimizer might purposefully act in ways that cause gradient descent to update (or not) it in some specific way is called gradient hacking.

Definition: Gradient Hacking

"Gradient hacking" describes the phenomenon wherein a deceptively aligned mesa-optimizer might be able to purposefully act in ways which cause gradient descent to update it in a particular way. - Evan Hubinger (2019) “Gradient hacking”

7.4.1 PREREQUISITES

Risks From Learned Optimization lists three prerequisites for an AI to become deceptively aligned. These conditions must be fulfilled during the training phase for an AI to develop internal goals, as well as the ability and desire to deceive the training process:

Situational awareness

The model must have a good enough understanding of the world to understand what humans “really want” and how it is different from what the AI wants (i.e. an understanding of the base objective). This is because in order to be successfully deceptive, the model still has to give the humans what they want during training. Additionally, it needs an understanding of the fact that it is part of a training process and some understanding of the technical details of how stochastic gradient descent (SGD) works. As an example, an agent tasked with forecasting stock performance in the tech sector might be given data about the entire world, including current AI technology in order to improve its performance. This information might also allow it to learn about its own situation. There is a lot of ongoing research and formalizing and detecting situational awareness in models.

Long-term planning

A model might be explicitly encouraged during training to acquire long-term planning capabilities to optimize its base objective. This might be, for example, because it was trained to take action in an environment where actions can have permanent consequences, or because of reasoning abilities. The possibility of permanently altering its environment means that the model has to consider the implications of its actions across longer time periods and training update cycles. Once it gains significant capabilities in long-term planning, it can use this ability to optimize for its own mesa-objective.

How could a model tell if it is training or deployment? RSA-2048

Expectation of distribution shift

The model must expect that at some point in the future, the threat of objective modification will go away. This means that the model thinks that training either has to end at some point, or that the trainer (SGD) does not need to update the behavior of the model because it is already performing well enough on the given task.

There are some dynamics of the gradient descent training process that might make the emergence of prerequisites like situational awareness and long term planning more or less likely. The next couple of sections explain machine learning training dynamics like inductive biases and path dependence. The subsequent section then uses this understanding to make an argument for the likelihood of deceptive alignment.

7.4.2 INDUCTIVE PRIORS

Inductive biases represent the constraints and influence how and what ML models learn. An inductive prior represents the initial state before any evidence has been observed. Here are some examples:

- **Complexity Bias**: It might be the case that the ML process prefers simpler models over more complex ones. The intuition behind this is Occam's razor, which suggests that the simplest explanation (or in this case, model) is often the “correct” one.
- **Smoothness Bias**: It could also be possible that smoother target functions are easier to find in search space than those that are erratic. In other words, small changes in the input should not produce large changes in the output.
- **Speed Bias**: Similarly, it could be the case that functions that can be computed quickly are more desirable.

There are many more inductive biases that the current machine learning process might have. The above are only a couple of examples. So depending on the inductive bias, gradient descent might just always simply find either the simplest, fastest, or smoothest model that fits the data. Which means that given the same dataset, all paths converge on essentially the same generalization.

Both inductive biases and path dependence can significantly influence the learning process and the final outcome of a machine learning model. A model's inductive priors can set it on a certain 'path', and then path dependence can further shape how the model learns and adapts as it travels down that path.

7.4.3 PATH DEPENDENCE

Path dependence refers to the update path that gradient descent follows through parameter space.

Path dependence studies the effects of changes in the training procedure on the path that gradient descent follows while searching for the optimal learned algorithm. This includes things like the sequence of training data, the order of training, etc…

High path dependence means that significantly different learned models emerge at the end of the training process. **Low path dependence** means that similar learned models emerge across multiple training runs with the same model on the same data.

This means that not only the emergence of mesa-optimizers but also overall levels of alignment could be sensitive to changes in the training procedure, which could make it harder to implement and enforce exact training procedures across different labs.

High path dependence

If gradient descent is highly path-dependent, then that would mean that different training runs can converge to very different models. This essentially means that the learned algorithm is very sensitive to the particular parameter update path that is taken through model space. To understand the types of expected learned algorithms, the path itself must also be understood.

There is some empirical evidence to suggest that many ML models might have high path dependence. The paper "BERTs of a feather do not generalize together" showed that different versions of the fine-tuned BERT models resulted in differing generalizations on downstream tasks, even with the same training data. This sort of path dependence is especially prevalent in reinforcement learning (RL) models, where an identical setup can be run multiple times, sometimes getting good performance and sometimes terrible performance.

Low path dependence

In this type of process, similar training processes converge to essentially the same simple solution, regardless of early training dynamics. In this case, inductive priors play a much larger role in what type of algorithm you might end up with.

Ajeya Cotra provided a simple example of low path dependence in her post "Why AI alignment could be hard with modern deep learning". The AI model was trained to be a shape detector. She defined a "thneeb" as one of the two shapes shown in the image below. Multiple models are trained to distinguish between two distinct types of shapes.

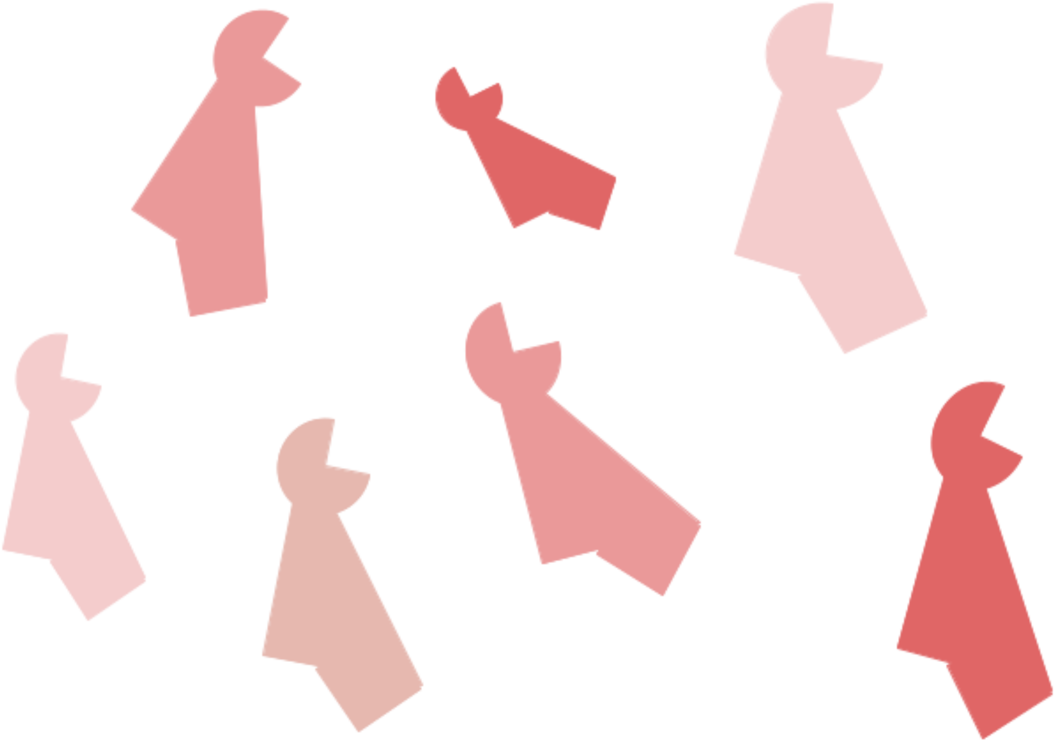


Figure: Thneeb

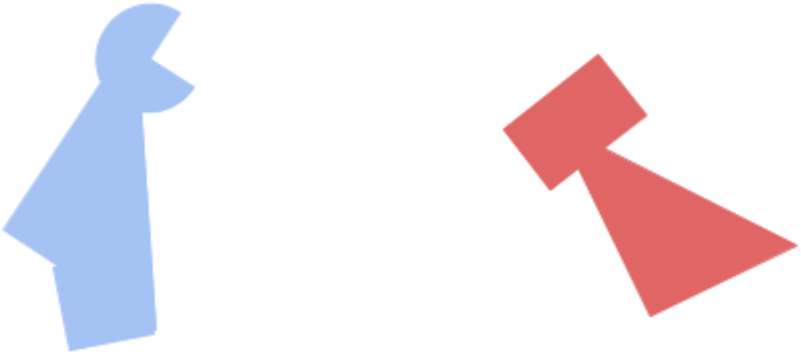


Figure: Thneebs with colors changed

Source: Cotra, Ajeya (Sep 2021) Why AI alignment could be hard with modern deep learning

However, in addition to the shapes in each cluster being similar, one shape was always blue, and the other was always red. During training, the model performed well at recognizing the shapes. During testing, the colors were swapped and it turned out that the model was only looking at the colors and not considering the shapes. On running the training process multiple times with random initialization points, it might be expected that at least one of the times the ML model generalizes to look for shapes instead of looking for colors. However, the paper "ImageNet-trained CNNs are biased towards texture" found that neural networks (or gradient descent) are consistently biased towards colors over shapes when given the choice.

Overall, there is no consensus on whether advanced machine learning models have high or low path dependence since there is evidence for both. The next section explores the likelihood of deceptively aligned mesa-optimizers emerging in both low and high-path dependent training scenarios.

7.4.4 DECEPTION =/= DECEPTIVE ALIGNMENT

⚠ **This section is still being written and is considered a work in progress.**

https://www.lesswrong.com/posts/a392MCzsGXAZP5KaS/deceptive-ai-deceptively-aligned-ai

7.4.5 LIKELIHOOD

Since the objective is to analyze the emergence of deceptive alignment, this section assumes the emergence of a mesa-optimizer during training. The focus is more on "which type" of mesa-optimizer (aligned, corrigible, or deceptive) is most likely to be found using current ML training methods.

A couple of further assumptions: First, there is a relatively complex base objective such as human values. Second, training environments are diverse enough with enough situations to get the models to robustly understand what humans want.

During training, ML models always start with a seemingly random proxy objective. To improve performance Gradient descent can do two things:

- **Better Proxies**: The model can improve its understanding of the objective
- **Better World Models**: The model can improve its ability to act on its existing understanding of the objective

High Path Dependence Scenario

If gradient descent is highly path-dependent, then depending on the path followed either the world model or the proxy could be perfected first. Which indicates that there could be the following different types of scenarios:

1. If the proxy objective is perfected to match the base objective before the world model is perfected, then the model is internally aligned.
2. If the world model is perfected before the proxy objective, then the world model is also perfect enough to simulate gradient descent. Since the world model is perfect the base objective also exists within the world model. In which case, one of two things can happen:
3. The mesa-optimizer 'throws out' the proxy objective and replaces it with a pointer to gradient descent's base objective.
4. **T**he mesa-optimizer just makes the current proxy objective a long-term goal (crystallizes it), which means that the mesa-optimizer is now free to maximize the base objective.

In all three of these cases, at the end of training gradient descent returns a model that optimizes the base objective. All three are functionally indistinguishable.

Low Path Dependence Scenarios

If training has low path dependence then the inductive priors come into play.

The simplicity inductive bias is an analysis of the algorithm's Kolmogorov complexity. In all three - internally, corrigibly, and deceptively aligned types of models the world model is exactly the same. The main difference between the models is the mesa objective, i.e. the complexity of the objective given to the model to learn and optimize. The simplest internally aligned mesa-objective is the base objective. So for internal alignment, the complexity class is the full complexity of the base objective. The simplest corrigibly aligned mesa-objective is a specification for how to point to the base objective. The simplest deceptively aligned mesa-objective is literally any long-term objective. So as far as deceptively aligned AI is concerned, the most complexity cost paid is just the simplest long-term objective that gradient descent can find. In other words, the deceptively aligned one has the lowest complexity cost out of the three alignment scenarios, assuming a relatively complex training objective. All of the other model classes scale with the complexity of the training objective, whereas the complexity of the deceptively aligned model is essentially fixed. Therefore the more complex the base objective (e.g. human values) the more deceptive alignment wins in the complexity race. Overall, the analysis results in the conclusion that the number of internally aligned AIs is less than the number of corrigibly aligned AIs which is less than the number of deceptively aligned AIs:

(internally aligned AIs) < #(corrigibly aligned AIs) < #(deceptively aligned AIs) Random sampling from a distribution of learned optimizers means there are more ways to get a deceptively aligned AI than the other two.

The speed inductive bias thankfully falls in favor of corrigibly and internally aligned models. The deceptively aligned model has to perform extra steps of 'thinking'. For example, it has to realize that caring about the mesa-objective in the long term means that it should optimize the training objective temporarily to get a higher reward on your internal objective eventually. The internal and corrigibly aligned models however don't have to do this extra thinking and can just directly optimize what they care for which is the base objective. So in terms of time taken for each model class to output the correct answer:

t(internally aligned) ~= t(corrigibly aligned) < t(deceptively aligned)