Approval-maximizing representations

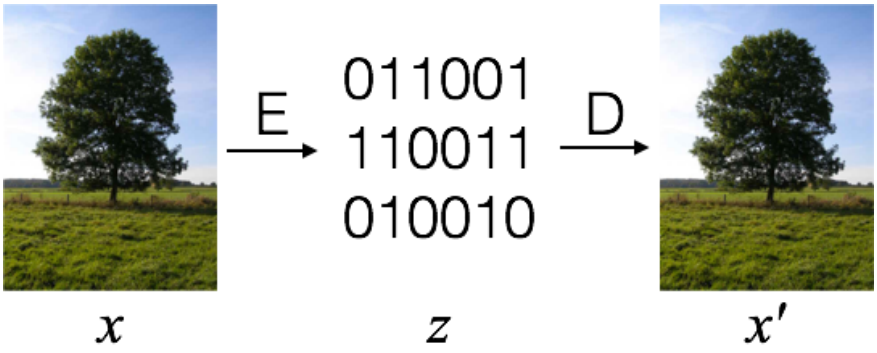
Paul Christiano Follow
Jun 17, 2017 · 7 min read

(*Follow-up to learning representations.*)

A long-lived AI will consume a lot of data, and needs to make hard choices about what to store and how to represent it. This is problematic for an act-based agent, where these decisions must be made or evaluated by an overseer.

We might wonder: can act-based agents ever learn to use representations that are incomprehensible to humans? If they can't, it would probably prevent them from achieving state-of-the-art performance.

As a warm-up, consider the case where we observe an image, and want to store a compressed version which we can recall on a future day. That is, we want to learn an encoder E : (image) → (code) and a decoder D : (code) → (image).



Schematic of an autoencoder

How should we train these maps?

Even if the overseer can't understand the code at all, we can train the encoder and decoder jointly. That is, we start from an image x, obtain the reconstruction $x' = D(E(x))$, and ask: "is x' a good reconstruction of x?"

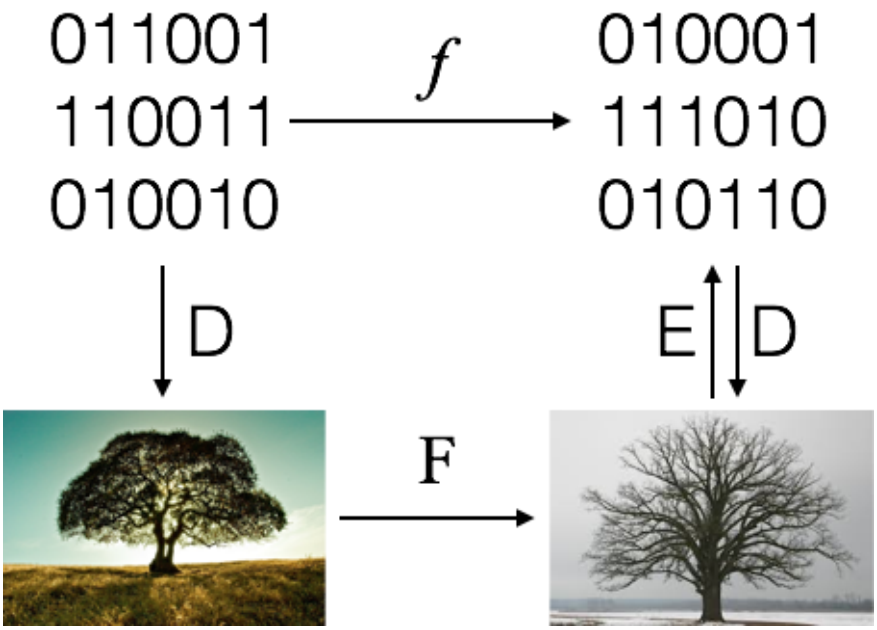
The overseer's responses define a loss function $L(x, x')$, and we can then train D and E to minimize this loss, subject to a constraint on

the size of the representation z. Such an autoencoder can learn to throw out details that the overseer doesn't expect to be useful.

Manipulating representations

Now suppose that we want to learn to *manipulate* codes. For example, I may have a target transformation on natural inputs like "change the season from winter to summer." I'd like to train an agent to implement the analogous transformation on codes.

That is, suppose that the overseer has a map F : (natural input) → (natural output), and an approval function L : (natural input) × (natural output) → [0, 1]. This is the data that we need to train an agent to manipulate natural inputs using imitation+RL.



By following D, then having the overseer apply F, then applying E, we can produce training samples for f. Similarly, by applying D to both inputs and outputs, we can elicit the overseer's feedback on f.

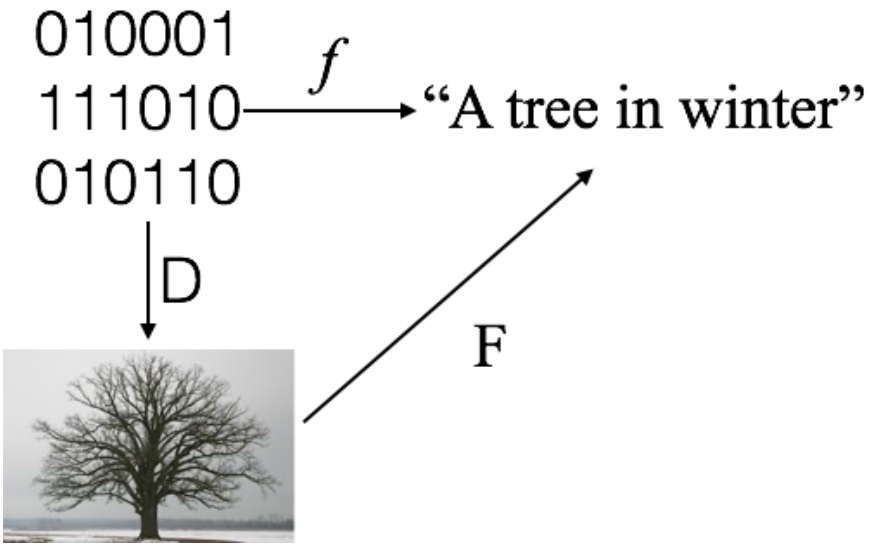
If we have an encoder/decoder, we can use them together with F and L to train a map on codes. To get examples we can evaluate E(F(D(z))). To get evaluations, we can compute L(D(z), D(f(z))).

So the overseer has lost nothing by working with these representations—they are able to provide feedback as well as if they understood the representations natively.

We can also learn representations which are computationally useful in addition to being compact, by training f jointly with the

encoder/decoder. A more convenient representation will make it easier to learn *f*.

At the end of the day, our system also interacts with the world by taking natural inputs (e.g. instructions in natural language, or images from a camera) and producing natural outputs (e.g. utterances, or images displayed on a screen).



f(z) ~ F(D(z)), which we can use both to sample and evaluate input/output pairs.

We can train these functions in exactly the same way, by translating to/from codes using our encoder or decoder.

If we were willing to train our system entirely end-to-end then we don't need to deal with the encoder/decoder at all and could just demonstrate/evaluate the natural inputs and outputs. The encoder/decoder are intended to allow the overseer to evaluate individual operations on codes, so that they don't need to evaluate the entire process from start to finish. This is important because our agents might perform very long sequences of operations, which won't be practical to train end-to-end. For example, an agent might want to use an observation months after making it.

Iterative encoding

In general, transforming an efficient representation into a human-comprehensible representation will increase its size.

For example, a comprehensible representation might be a description in English, while an efficient representation uses a richer language containing concepts that humans don't understand. Translating the

rich language into English may greatly increase its size—perhaps a single word of the rich language can only be explained with many paragraphs of English text.

In general, unpacking an efficient representation could lead to exponentially large comprehensible representations, which would render the schemes from the last section impractical.

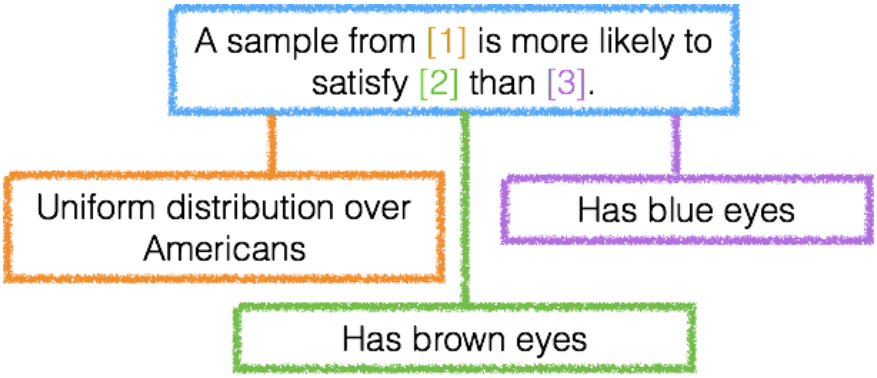
We can potentially address this problem by working with a sequence of increasingly complex representations.

Compound representations

The first ingredient is building a large representation out of smaller parts.

We'll start with some space of "simple" representations S, such as snippets of English text that can be read and understood in 5 seconds. We'd like to build more complex representations out of S.

This could be approached in many equally good ways, but I'll use the idea of *messages* defined in meta-execution.



A compound representation composed out of simpler representations.

A message over S consists of two parts:

- An element of S (the "head")
- A list of additional messages over S (the "arguments"), that can be referenced by the head.

The semantics of messages are straightforward, and hopefully the example above makes it clear.

The *size* of a finite message is 1 plus the sum of the sizes of all of its arguments. Write M(S) for the set of messages over S of size at most 1000.

M(S) is a much bigger set than S, and can generally represent much more complex concepts.

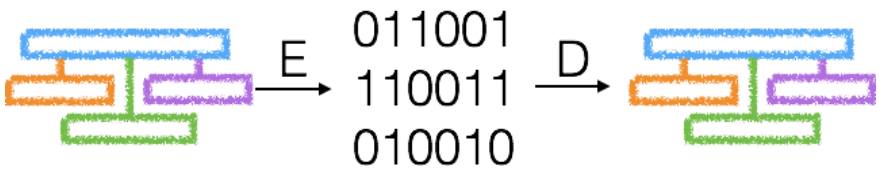
Compressing representations

By combining compound representations with representation learning, we can potentially construct a hierarchy of increasingly efficient representations.

Let S be some initial representation which is comprehensible to the human overseer H. An agent A trained by H would naturally operate on S.

Now we'd like to train agent A^+ which is smarter than A and operates on a more efficient representation S^+.

If we use an amplification scheme like meta-execution, then we can turn A into a smarter agent H^A which operates naturally on M(S). We can then use H^A to train an autoencoder which compresses M(S) into a new representation S^+:



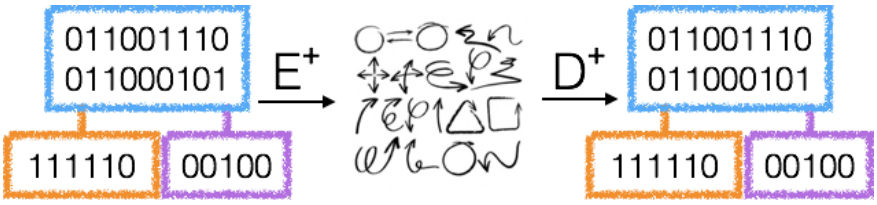
An autoencoder which translates compound representations into a more efficient encoding. The empty boxes would be filled in with elements of S, and the binary code belongs to S^+.

In general this encoder can't preserve all of the information in every compound representation. But it can preserve the most important details, and may be able to substantially compress the representation.

We can train this autoencoder jointly with a new agent A^+ who operates on S^+, in order to obtain representations that are computationally convenient. With the encoder in hand, H^A can effectively oversee A^+.

Iteration

Now we have an agent A^+ which operates on a more efficient representation S^+. We can repeat the process: H^{A^+} can train a new agent A^{++} which operates on a still more efficient representation S^{++}:



An autoencoder which translates $M(S^+)$ into a still-more-efficient encoding S^{++}.

In this picture, E^+ takes as input a message over S^+ (represented as binary) and outputs a symbol in S^{++} (represented as squiggles).

H^{A^+} is able to operate on messages over S^+ since it is able to call A^+ many times, and so it can train E^+ and D^+. And with the encoder in hand, H^{A^+} can train a new agent A^{++} which operates on S^{++}.

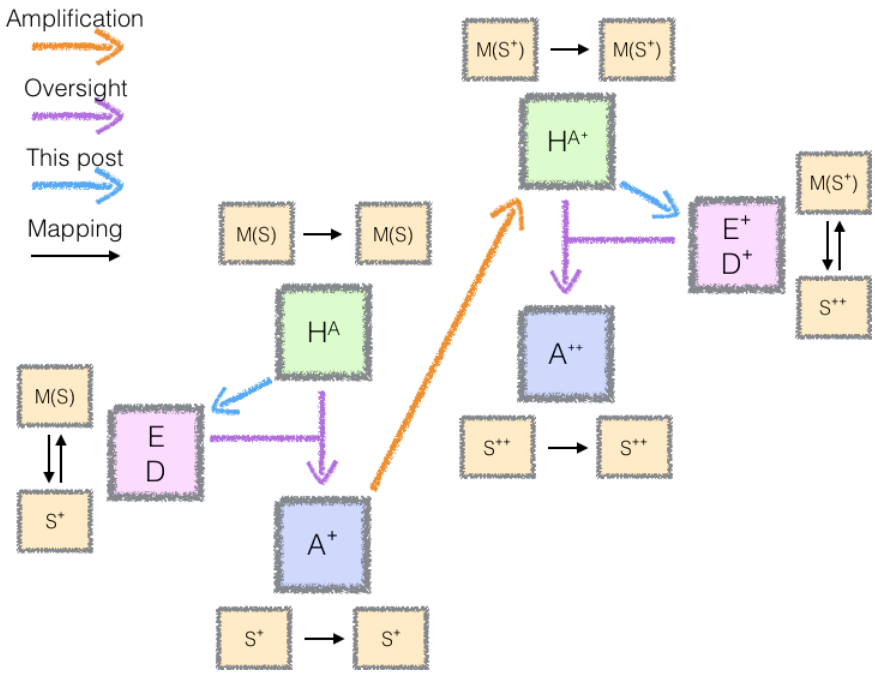
Note that while it is possible to convert S^{++} back to a message over S, the size may grow considerably. An element of S^{++} can be unpacked to a message of size 1000 over S^+. Each of these thousand elements of S^+ can be unpacked to a message of size 1000 over S. And so an element of S^{++} would unpack into a message of size 1000000 over S. This size will keep growing exponentially as we iterate.

Putting it all together

I've been somewhat cavalier when talking about bootstrapping schemes for aligned RL, and haven't discussed the representations they would use. I've swept this under the rug by assuming that the resulting agents operate in an episodic environment with a well-defined beginning and end and that all information stored between episodes is human-comprehensible.

The primary purpose of this post is to patch up these oversights, and to move towards removing the episodic assumption altogether.

So what does it look like if we combine this system with this bootstrapping scheme, putting together the steps from the preceding sections?



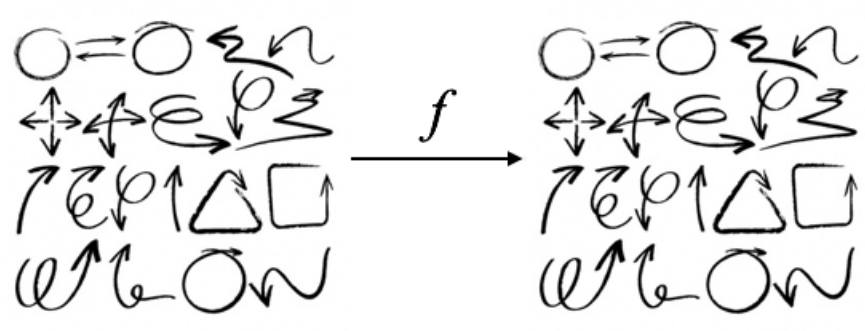
It's a mouthful.

This is a copy of the diagram from the previous post, but with the following changes:

- I've annotated each policy with its domain and range. As before, S, S^+ and S^{++} are increasingly efficient representations, and M(X) is the space of small messages over X. For convenience, I'm assuming that each of S, S^+, S^{++} also contains fixed representations of the object-level inputs and outputs of our agents.
- I've inserted encoders E and E^+, and decoders D and D^+. These are trained using the mechanism described in this post. They are needed to provide oversight to A^+ and A^{++}, since H^A and H^{A^+} aren't capable of directly understanding S^+ or S^{++}.

Conclusion

Ultimately, efficient AI systems will act on compact representations which will be incomprehensible to humans. If we want to build act-based agents, we need to connect these representations to something that the overseer can understand.



This might be achievable by training a hierarchy of increasingly efficient representations in parallel with training a hierarchy of increasingly competent agents. We can learn richer representations by training an autoencoder to compress compound representations built up of simpler pieces.

It is completely unclear if this proposal is realistic, and this post hasn't touched on any of the practical difficulties. But it suggests to me that there isn't an inherent philosophical problem with act-based training of agents to operate on efficient, incomprehensible representations.