

The Big Blob of Compute Hypothesis

1. Introduction

I've considered several times writing up my overall perspective on AGI safety. The main reason I haven't done so so far is that my view of the problem is pretty high-level -- it's from a different genre than the kind of "framework-y" agendas that Paul and Eliezer have written, and in fact a big part of it is arguing that such frameworks operate on the wrong level of abstraction (although that said, Paul's framework is broadly compatible with things that I think, whereas Eliezer's framework is broadly incompatible). However, then I realized that the core belief behind my safety perspective is *also* central to my perspective on AGI timelines, so it seemed particularly important to write it up. Some caveats: this is not a precise hypothesis or one I'm super confident in, and I have additional, more specific, reasons for holding both my views on AGI timelines and my views on AGI safety. It's more like a distillation of my high-level intuitions (informed by experience) about how intelligence probably works, that causes me to expect things to go a certain way by default, and makes me *a priori* highly skeptical of research directions that don't confront or don't seem to fully understand this picture. I'll describe the view first in terms of just AI itself, and then later go into the (subtle) implications for safety.

A big division between people who are bullish vs bearish on AGI timelines is the question of whether raw compute or new algorithms are more important to AI progress. I'm definitely closer to the "raw compute" side, but I also think the question is imprecisely stated -- what counts as an algorithm, how exactly does the raw compute have to be applied, and what about things that aren't either algorithm or compute (like design of environments)? Also, is lots of compute merely *sufficient* to get to AGI (i.e. it's one path among several), or is it actually somehow *necessary*?

My picture of the situation is what I call the *Big Blob of Compute* (BBOC) hypothesis. It says that creating any given intelligent behavior is mostly about providing a large, minimally structured mass of computational capacity, and then giving it shape and form via interactions with a rich environment and a training process that drives it towards behavior appropriate for the task at hand. These high-level knobs -- what is the training signal (or signals)?, what is the nature of the environment the agent interacts with?, how much experience is it exposed to? -- constitute the most effective ways of controlling how well an intelligent agent performs a given task and how specifically it performs it. Once these knobs are set, then learning to perform the task is mostly about the agent having enough compute and storage capacity to learn the required cognitive skills.

A few algorithmic settings do matter, but they tend to fall into a few very simple categories -- they exploit simple symmetries of the world, they help to regularize and condition the compute, or they make the shape of the computational graph smoother and less awkward. More complicated algorithmic ideas rarely help, and in particular supposedly "fundamental"

differences in algorithms (I'll give examples below, like bayesian vs non-bayesian methods, model-based vs model-free RL, or even ML vs non-ML methods) usually end up mostly reindexing or redescribing the same underlying computations, shuffling around exactly the same processing in a kind of shell game, all while giving the surface appearance of doing something substantively different. Also, algorithms which attempt to rigidly prescribe what the compute should do, or how it should be divided up among alleged (according to humans) "pieces" of the task, tend to do particularly poorly, often leading to much worse results compared to an unstructured compute blob of the same size. More broadly, algorithms matter much more in a negative sense than a positive sense -- it's possible to come up with horrible architectures that totally block the flow of compute, but it's really hard to beat the unstructured blob by all that much, and it's also really hard (perhaps impossible) to use algorithmic changes to get the agent to do anything other than what's implied by its environment and training process.

This picture informs my view of both AI timelines and AI safety. Since the summary above is somewhat abstract, I'll first give (section 2) a more detailed formulation of BBOC, and (section 3) examples of where it's been true or where I predict it's going to be true. Then I'll (section 4) describe the perspective on safety that it suggests, and (section 5) describe why that leads to skepticism of MIRI's HRAD agenda and mild skepticism of "framework"-style AI safety plans in general. Finally, I'll describe (section 6) what I think a BBOC-compatible approach to safety should look like, and how that relates to what the safety team is currently doing at OpenAI.

2. More Detailed Formulation of BBOC

Suppose you want to learn and then repeatedly execute some task, like recognizing images, playing video games, doing science, proving theorems, imitating a human, or even the skill of learning new things quickly. How do you do this? The BBOC hypothesis says that you should worry mostly about the following 7 things; everything else usually has a minor impact or is operating at too low a level of abstraction to be properly general:

1. **Compute:** You need enough computation to learn the task at hand. You need the computation to flow smoothly and freely between observations of the environment and the agent's actions. It should be relatively unstructured and responsive to the training process; aside from that its exact form is secondary.
2. **Parameters:** The agent needs a mechanism to persistently store information that it learns about the general task during training, and transiently store information that it learns about an instance of the task during execution. This information can be implicit or explicit, directly stored observations or the parameters of a statistical model. (In simple deep neural nets the persistent and transient information are the weights and the activations; for something like evolution it's the genome of a class of organisms and the synaptic weights of a particular brain.) If you don't have enough parameters, you won't learn the task well.

3. **Quantity of experience:** The agent needs enough experience acting in the environment where the task is to be performed. I use the word experience instead of data because static data is a special case of interaction with a dynamic environment. Experience can to some extent be traded off against compute if the agent can internally simulate some aspects of experience.
4. **Distribution of experience:** The distribution of experience during learning must be such that the simplest way to do well on the training tasks also does well on an instance of the final task. One way to achieve this is if training tasks instances are drawn i.i.d. from the same distribution as test task instances, but it also works if the training tasks vary in such a way that it takes less bits to find the high-level commonalities between them (say the laws of physics, or some other small description length program) than it does to separately solve each task. Then, even if the test task is very different from the training tasks, the agent will do well as long as the high-level commonalities are preserved. Separately from this, it may be necessary for the training tasks to be simpler than the final task, or to increase smoothly in difficulty.
5. **Normalization and conditioning:** Freely flowing numerical computation can very easily have numerical stability or conditioning issues. Many of the “algorithmic” innovations in the last 5 years of AI have actually been simple methods for normalizing and conditioning numerical computation: consider [Adam](#), [BatchNormalization](#), or [ResNets](#). These methods basically just say that numerical quantities that occur in some aspect of the computation should be normalized to 1 or reparameterized as a diff from 1. This simple “flow control” has often outperformed much more sophisticated ideas either within or outside deep learning.
6. **Symmetries and shape:** Another huge chunk of the most impactful algorithmic innovations has been the use of symmetry to decrease the number of parameters in models, and thus make them learn much faster. Intelligence only works at all because we live in a low entropy world; one way the low entropy manifests is symmetries and sparsity in the physical world. For example, convolutional nets take advantage of spatial translation invariance; RNNs take advantage of time translation invariance (memory architectures might be a future candidate in that they take advantage of general sparsity). We could likely learn the same tasks without these symmetries, but it would take a lot more compute and parameters (which would then eventually discover the symmetries on their own). In a similar vein, the shape of the computation can matter -- we don't want information to bottleneck at a very small layer, or be forced to compress observations before analyzing them. The shape of the computation should be flexible and sensible. As tasks get more general, symmetries may become less important, as they may not hold across tasks and we may also be able to train meta-learning architectures that experiment with various symmetries at the object level.
7. **Training target:** The computation needs to be driven by the correct evaluative mechanism for the task at hand, which can be a loss function, a reward, a prediction error, a GAN loss, a fitness function, a human evaluation, an extended human interaction, or a process for combining any of the above. You will learn whatever the training process asymptotically incentivizes; if you have the wrong training process, you'll

learn the wrong task. This isn't just a comment on safety: many deep learning papers today learn the wrong task: for example, dialog systems that attempt to match human responses in a conversation (the goal of human conversation isn't to match the responses of some other human, but to accomplish whatever the human's goal is in the conversation), non-metalearning systems that train on one task and then apply the results to another (the agent was not trained on the task of transferring to another task, therefore it won't be optimal at that). Note that the "training target" is not necessarily a single objective function or reward function; it can be a complicated process with several parts that together drive towards some asymptotic condition. For example, the training target in RL from human feedback comes from combining two different learning processes and is something like "learn to act in the environment in such a way that short clips of your actions would be preferred by a human relative to short clips of other actions you could take".

BBOC says something like "if you get 1-6 right, you will probably learn the behavior that 7 specifies if it's learnable in a computationally tractable way. Things not included in 1-7 have a much lower likelihood of being important, although there are exceptions."

A related concept to BBOC (that may help with intuition) is what I call the "snowflake model" of intelligence. If you looked at the fractal structure of a snowflake, you might think that whoever made it did something impossibly intricate and difficult, but that building it piece by piece must somehow be possible because someone did it. In fact, both statements are false: the way to make a snowflake is not to think in terms of its pieces but to know the laws of physics (training target), have enough raw material (compute) and a large enough chamber (parameters), set the temperature, pressure, and humidity correctly (normalization, shape), and wait for long enough (quantity of experience). Furthermore, this is your only way to make snowflakes and your only leverage over their shape; trying to piece together a single one from little bits of ice is basically hopeless. For snowflakes you might call this the "big blob of snow" theory.

Ilya Sutskever expresses BBOC as "networks want to learn".

It's worth noting some exceptions or apparent exceptions. When training on a narrow version of a task (like playing a single atari game, rather than playing any video game), it's often possible to hand-engineer a method for that task in particular, leading to apparent large algorithmic improvements and more prescribed structure to the compute, but this comes at the cost of brittleness and poor generalization to broader tasks (e.g. consider the flood of papers proposing small tweaks to RL on atari). This phenomenon of (often unknowingly) overfitting to narrow tasks creates the illusion of successful algorithmic innovation, while *also* making our agents appear brittle and overcomplicated, *and* masking the simple relationship between compute and results. I believe this gives a very distorted view of the field for people who mainly pay attention to small tasks and expect breakthroughs to appear there first. By contrast, experience with

huge projects seems to cultivate the BBOC intuition¹, and I also believe that BBOC will become more obvious as we move to more general tasks (which will be algorithmically *simpler* to learn than more narrow tasks).

An extreme case of the paragraph above is the case of tasks that are so narrow that you don't need any parameters at all and a human can simply write a short program that solves them (or experiment with a small range of such programs). Examples of this might be sorting, graph navigation, or alpha-beta pruning. Here the method solves the narrow task really well, can be much more or less efficient depending on the algorithm you use, and generalizes perfectly within instances of that narrow task (for example, sorting a larger list of numbers), but has zero ability to generalize outside that task (i.e. responding well when you give it an algorithmic task other than sorting). Basically all AI work that came before ML was of this form -- pre-ML AI was mostly just programming focused on certain selected domains. This history obscures BBOC and makes it look like huge gains from algorithms are possible and that the right algorithms give perfect generalization. In reality, however, generalizing just a bit beyond these single algorithmic tasks requires searching over programs, which will inevitably be heuristic, will inevitably drag in learning/ML/parameters, and if the task is general enough (e.g. solve an arbitrary programming interview question) will require a lot of compute and will follow the normal rules of BBOC.

As written, much of BBOC's description may read like a common sense guide to designing deep learning systems, but most people have not thought carefully about how far this extends or what it implies about intelligence. Many things that people commonly say about deep learning, both within the field and outside it, are strongly contradicted by BBOC, and start to sound silly if you really believe BBOC. Below I give some examples about what matters and what doesn't in AI training, that helps to make clear how surprising and far-reaching it is.

3. Evidence and Examples of BBOC

Here are some things I've observed over 3 years of training AI systems and many years of studying the brain, that are either evidence for BBOC, or implications/predictions of BBOC:

- The classic (and best known) example of BBOC is the use of convolutional neural nets for vision. In the years before 2012 there were elaborate attempts to design edge detectors, contour integration systems, segmentation systems, foreground-background analyzers, pose estimators, and so on, which people then tried to put together to make vision systems. These systems failed because they did not use enough compute (1), did not have enough parameters (2), organized the computation too rigidly (6), and did not have the right objective function (the pieces were trained on e.g. edge detection rather

¹ My first job in AI was working on what turned out to be what I now believe was the single largest use of compute in one training process up to that time, [Deep Speech 2](#), which used about 0.3 petaflop-days per trained model in 2015.

than identification of the final object) (7). CNN's fixed all these problems. If you keep the elaborate systems but fix (1), (2), and (7), they can do passably on imagenet, but they are still inefficient, complicated, and generalize poorly.

- Basically my entire experience with training AI systems lines up with the CNN case -- use the simplest architecture and algorithm you can, and get (1)-(7) right, and you'll do well. If you don't have enough compute, parameters, or data, nothing can save you. Speech, language modeling, translation, reinforcement learning, generative models, soon robotics -- they all seem to follow this pattern. It takes only a few words to state this fact, but the weight of experience, of training hundreds of models, trying hundreds of things, and seeing what works and what doesn't, is hard to overstate.
- When I was at Google I was involved in a project to see if neural nets are really essential to deep learning. We tried to replace the matrix+nonlinearity in each layer with some other kind of machine learning model (I am not allowed to say exactly what), basically making a "deep <some other ML technique>". We found that these models worked but consistently performed slightly worse than similarly-sized neural nets. Eventually I discovered that any of these custom layers could form a standard neural-net layer with a subset of its parameters if it wanted to, and that is what it had chosen to do; basically it had reparameterized the model we gave it to internally and surreptitiously form a neural net. In other words, the different layer literally caused rearrangement of the same computation that a neural net would have done. On one hand, this tells us that deep neural nets are pretty effective models (other models want to become them), on the other hand, it tells us that other models can work too and there's nothing magic about "deep learning"; it's more that certain computations really want to occur and will shape themselves around whatever type of model you give them.
- In RL textbooks there is a distinction between model-based RL and model-free RL. The first has an environment model that makes predictions and also a policy, the second has just a policy. In theory model-based RL is a bit like system 2 in that it can think ahead and plan whereas model-free RL is like system 1. In practice model-free RL tends to do some planning internally, and we're starting to suspect that if we simply shaped model-free policies differently, say allowing them to iterate themselves for many steps before taking an action, that they would be able to (implicitly) do planning at least as well as model-based systems. Meanwhile actual model-based RL systems perform relatively poorly, perhaps because we're artificially forcing computation to take place in 2 disconnected blocks. The point is, the computational activity of planning is important, but whether or not that activity occurs has more to do with the shape of the blob of computation than with the algorithm we're supposedly using or which neural net is supposed to do what. "Model-free" systems learn a model internally when their environment and objective function require them to, and if it's sometimes a deficient model that may just be because we don't tend to shape model-free networks so as to give them enough internal recurrence to implicitly do model-like computations.
- Similarly, ordinary RL is said to have issues with long time horizons, say planning over 1 million timesteps. Hierarchical RL is supposed to deal with this; the idea is to have goals and subgoals, and different policies set the goals at different timescales. Unfortunately

so far hierarchical RL hasn't been able to set goals and subgoals much better than they can be learned implicitly within a normal RL algorithm; there's a tendency for the hierarchy to just collapse into approximately what a single (deeper) neural-net would have done. It's possible we'll eventually make hierarchical RL work, but for now it seems to be yet another example of compute having its own idea about what it wants to do and routing around the structure imposed on it by a supposedly important algorithm. Meanwhile, ordinary RL algorithms are having surprising success on long time horizons when we make the policies really large, sometimes solving tasks with only 1 reward every 10,000 timesteps. We really will eventually need to represent hierarchical structure within our RL policies, but that may happen implicitly rather than through the techniques we've labeled "hierarchical RL".

- LSTM's are supposedly better than simple RNN's because they can store state for longer -- one of the few real algorithmic innovations in deep learning. Yet there's now evidence that simple RNN's perform as well as LSTMs when properly tuned; the only advantage of LSTMs is that they work well for a wider range of hyperparameters. In fact, holding constant the number of parameters and the amount of compute, different recurrent architectures (RNN, LSTM, GRU) seem to have eerily similar capacity and performance.
- When current neural nets don't generalize well people often talk about this as if it's some kind of algorithmic defect, as if there's some yet-to-be-discovered algorithm somewhere that generalizes drastically better². But my experience is that neural nets generalize better when you simply expose them to a wider distribution of data that forces them to find a general solution rather than just memorizing a few special cases. For example, when training speech models I found that if you trained only on an American accent you would do poorly on many other accents. However if you train on 5 or 6 accents you will immediately do well on an unseen accent. What's probably going on is that for 1 or 2 accents it's easy to just memorize the peculiarities of each one, whereas as the number increases it becomes easier (and requires fewer bits) to identify the commonalities between them and store these, which leads to generalization. To summarize: it's not algorithms that lead to generalization, so much as the training setup and environment.
- I sometimes hear discussion, in both the academic ML world and in the AI safety world, that deep neural nets are missing something and that Bayesian methods would do better in some way (they'd represent uncertainty, be better calibrated, it would be easier for us to understand their beliefs, etc). But in fact neural nets already do represent uncertainty (with e.g. class probabilities), and could even output a full joint distribution rather than factored probabilities if they wanted to, just like idealized Bayesian methods. The problem is this would take exponentially more parameters, in both the case of neural nets and some idealized Bayesian model; both run into the same practical limitation. People also say that if you had a Bayesian model, you'd understand explicitly what propositions it believes and how much, but this has never made sense to me -- the hard

² I believe this comes from the experience with pre-ML AI systems, where picking the right algorithm meant you could generalize perfectly within a single algorithmic task, basically because this kind of "AI" was just programming. I discuss this a bit in the previous section.

part is figuring out how to carve the world up into “statements”, and this has to be learned by whatever structure, Bayesian or not, you use to hold and fill your model (for example, a Bayesian network or a neural net). Once you’ve done that, the beliefs held in the nodes of a Bayesian network are just as likely to be comprehensible or incomprehensible, calibrated or uncalibrated, as the activations inside a neural network, the latter of which appear for all the world to be holding beliefs and weighing evidence even though they’re not explicitly labeled as doing so. Finally, people also say that Bayesian models are better at generalization and avoiding overfitting. This is true in the ideal limit (since you’re modeling a distribution), but in practice the simplest ad hoc methods that deep learning has improvised to deal with overfitting, like dropout, can be shown to be equivalent, to a Bayesian approximation. Meanwhile, people have actually tried explicitly Bayesian neural nets (with uncertainty on the weights), but they don’t stand out as clearly superior in generalization or calibration; they behave similarly to methods like dropout, as you’d expect. Once again the key point here is: the algorithms per se matter less than they appear, and give the appearance of something different going on when in fact they often just relabel the same computational process.

- People seem very surprised about adversarial examples -- they say it means neural nets don’t really understand the objects they’re classifying, or are brittle in some way. But BBOC isn’t surprised by this: the classifiers weren’t trained on a distribution of experience that included images engineered in this way. In line with BBOC, the current best defense against adversarial examples is to train on them directly. The current second best defense (and the thing I think will ultimately be the solution) is to have a generative model that tells you what is or isn’t a natural image; this works because the generative model was trained with an objective function that tells it to distinguish what is on the data manifold versus what isn’t, whereas the classifier had no such objective function. BBOC says you get what you optimize for; what you don’t get is a human’s idea of what properties the system should have.
- News articles occasionally talk about how we need some alternative to backprop, as if we’re missing some key algorithmic insight, some new kind of update rule that will revolutionize everything we do and allow our networks to finally learn in a truly human-like way (“hebbian learning” is often mentioned). It’s quite possible we could find a new update rule (in fact I suspect many gradient-like updates would be comparably good) but I don’t expect any such change to have anything more than a very modest effect; navigating high-dimensional spaces is hard and empirically we’ve tried a huge number of things that end up doing only modestly better than the simple gradient. Ascribing this almost totemic importance to the update rule is an example of the kind of thinking BBOC is designed to counteract.
- For a while people were doing transfer learning by training a policy on a bunch of tasks and then trying to fine tune on a new task; people tried a bunch of complicated variants of this that all worked about equally poorly. This is actually the wrong training procedure (it doesn’t asymptotically lead to fast-adapting policies) and things started working somewhat better when people switched to the right objective function: training an initial policy optimized for its performance after fast adaptation (meaning, learning and

evaluation on the task of adapting quickly to a new environment). Again, the detailed algorithm didn't make much difference; what seems to help is having the right target.

- There are a lot of algorithms out there that attempt to get better exploration for reinforcement learning, or better intrinsic motivation, or that attempt to explore more safely, etc. A lot of smart people (including me) have worked on these problems, and come up with algorithms that seem to get improvements, but over time these have become a messy and unwieldy patchwork, somewhat reminiscent of the pre-CNN edge detectors. It's easy to think we're beyond that because these methods use end-to-end deep RL, but maybe we need to go up one level of abstraction, learning over many agent learning processes how to explore, how to motivate, and how to explore safely. We've seen early progress on this type of "metalearning", but the jury is still out on where things will ultimately go.
- What about non-ML systems, like search or reasoning systems? Aren't they doing something genuinely different, and wouldn't you expect them to generalize better because they contain a symbolic recipe, while ML systems don't? What I believe is going on here is a misunderstanding -- ML systems are perfectly capable of driving symbolic computation, as all the work in neural-net driven theorem proving shows, and they can also search over algorithms or programs based on how they fit the data (you can use neural nets to evolve code, though not well yet). In fact I think neural net systems exposed to a wide variety of environments with only high-level similarities (e.g. the laws of physics) will, if their shape and structure is organized in the right way, inevitably converge on parameters that implement simple programs (e.g. they will derive or represent the laws of physics) -- our brains did this despite being based on very fuzzy computation. Once that happens, neural nets will be searching over programs whereas non-ML methods represent fixed programs (or small families of programs, like tree search); at that point non-ML methods will just be doing a fixed, rigid subset of the computation done by ML methods³. So we can expect that ML systems will do much better than non-ML systems (even at symbolic tasks) while using more computation to do so, in line with BBOC. We can already see this with MCTS -- DeepMind has published some papers showing that learning to search can outperform fixed tree search algorithms, and it's only a matter of time until this is incorporated into e.g. AlphaGo.
- Consider human evolution. How did nature manage to discover and develop brains capable of uncovering the laws of physics, proving Fermat's last theorem, and composing symphonies? You can trace step by step how it happened, and give explanations for each step, yet the process as a whole is still surprising in its power. BBOC offers a simple high-level explanation -- enough compute, enough minimally structured brain memory, a broad enough distribution of environment experience and challenges, and a driving objective that involved playing agents off against each other were sufficient to make the result inevitable eventually.

³ This is also discussed in the section above: early AI work focused on fixed programs or small families of programs that solve specific narrow problems; calling this work "AI" obscures BBOC and erroneously leads us to think there's something different about symbolic computation.

Hopefully the above gives some flavor of why I see BBOC as so likely, despite seeming strange at first glance and not being obvious from looking at small-scale benchmarks.

Note that none of this implies that scaling up today's exact algorithms will lead to AGI. It does suggest that any algorithmic changes will likely be simple things that exploit symmetries or sparsity (good candidates might be memory-based models, or neural nets that grow in size over the course of training), or innovations in the training process, and that scale will be a huge part of the story. It also suggests that "alternatives to deep learning" are at very high risk of either not working or just recapitulating deep learning, and that algorithmic innovations said to introduce some desirable property should be treated with skepticism.

A striking feature of BBOC is that a research direction may look perfectly reasonable for a long time (visual edge detectors, model-based RL, hierarchical RL, exploration) and have many of the smartest people working on it, before it becomes clear that it's simply rearranging compute or trying to be too prescriptive about how to solve a problem. Over time, however, you develop an intuition for when an approach is "too handcrafted", similar to a programmer's intuition about when an abstraction is insufficiently general or modular. Model-based, hierarchical, and exploration are still controversial -- maybe we'll still get something out of them -- but they are starting to look increasingly suspicious, increasingly like the edge detectors of 2011.

A final note is that sometimes it's hard or impossible to take the BBOC-recommended approach to a problem right away. Sometimes you need to start with something lower-level on a narrower problem (e.g. start with ordinary deep RL on one video game before you try meta-RL on many video games) to get your bearings and understand how to design the higher-level algorithm. So narrower projects where you design lower-level architectures can be useful as stepping stones, something I'll come back to when I discuss my research strategy on AI safety (section 6).