Background

I have become a person of odd traditions, and one of them is: twice a year (in November, and again in May), I spend a month on what can really only be described as a machine learning writing marathon: reading a paper, and writing a 500-1000 word summary each day of the month. This is the third time. It was originally inspired by November's National Novel Writing Month, but last year I also added in May, because there had been such an interim buildup of papers I was excited to read. I'll be honest: more than anything else, I do this because it is a high-intensity way for me to learn a lot in a short period of time, and I love that. I've discovered that just deciding you have to do a thing, and you don't have a choice, is a kind of a superpower: it lets you leverage stubbornness to do things you couldn't do otherwise. But, I also love knowing that what I write has been valuable to other people trying to make their way in the world of scientific paper understanding, with all of its odd conventions and sub-par incentives towards clarity, which is why, in addition to posting my writing on shortscience, I also put together these full-month aggregations. There is some variance in quality here (I hope you'll be generous in recalling some of these were written after 11PM), but all in all, it's work I'm proud of, and am glad to have done.

Happy reading, and I hope you learn something interesting or useful here!
-Cody Wild

Language/Sequence Models

BERT: Language Model Pretraining

https://arxiv.org/abs/1810.04805

The last two years have seen a number of improvements in the field of language model pretraining, and BERT - Bidirectional Encoder Representations from Transformers - is the most recent entry into this canon. The general problem posed by language model pretraining is: can we leverage huge amounts of raw text, which aren't labeled for any specific classification task, to help us train better models for supervised language tasks (like translation, question answering, logical entailment, etc)? Mechanically, this works by either 1) training word embeddings and then using those embeddings as input feature representations for supervised models, or 2) treating the problem as a transfer learning problem, and fine-tune to a supervised task - similar to how you'd fine-tune a model trained on ImageNet by carrying over parameters, and then training on your new task. Even though the text we're learning on is strictly speaking unsupervised (lacking a supervised label), we need to design a task on which we calculate gradients in order to train our representations. For unsupervised language modeling, that task is typically structured as predicting a word in a sequence given prior words in that sequence. Intuitively, in order for a model to do a good job at predicting the word that comes next in a sentence, it needs to have learned patterns about language, both on grammatical and semantic levels. A notable change recently has been the shift from learning unconditional word vectors (where the word's representation is the same globally) to contextualized ones, where the representation of the word is dependent on the sentence context it's found in. All the baselines discussed here are of this second type.

The two main baselines that the BERT model compares itself to are OpenAI's GPT, and Peters et al's ELMo. The GPT model uses a self-attention-based Transformer architecture, going through each word in the sequence, and predicting the next word by calculating an attention-weighted representation of all prior words. (For those who aren't familiar, attention works by multiplying a "query" vector with every word in a variable-length sequence, and then putting the outputs of those multiplications into a softmax operator, which inherently gets you a weighting scheme that adds to one). ELMo uses models that gather context in both directions, but in a fairly simple way: it learns one deep LSTM that goes from left to right, predicting word k using words 0-k-1, and a second LSTM that goes from right to left, predicting word k using words k+1 onward. These two predictions are combined (literally: just summed together) to get a representation for the word at position k.

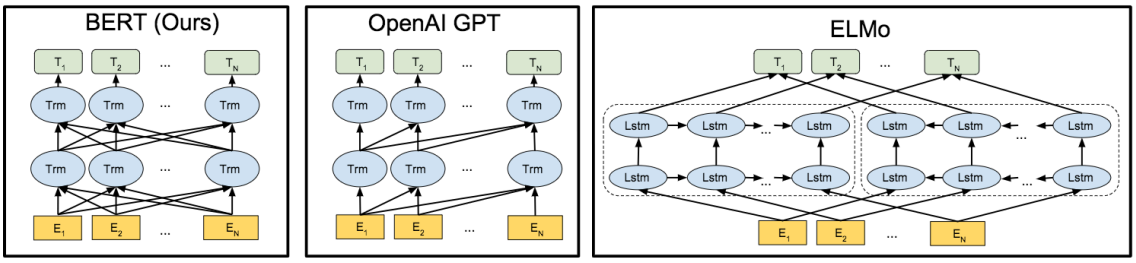


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

BERT differs from prior work in this area in several small ways, but one primary one: instead of representing a word using only information from words before it, or a simple sum of prior information and subsequent information, it uses the full context from before and after the word in each of its multiple layers. It also uses an attention-based Transformer structure, but instead of incorporating just prior context, it pulls in information from the full sentence. To allow for a model that actually uses both directions of context at a time in its unsupervised prediction task, the authors of BERT slightly changed the nature of that task: it replaces the word being predicted with the "mask" token, so that even with multiple layers of context aggregation on both sides, the model doesn't have any way of knowing what the token is. By contrast, if it weren't masked, after the first layer of context aggregation, the representations of other words in the sequence would incorporate information about the predicted word k, making it trivial, if another layer were applied on top of that first one, for the model to directly have access to the value it's trying to predict. This problem can either be solved by using multiple layers, each of which can only see prior context (like GPT), by learning fully separate L-R and R-L models, and combining them at the final layer (like ELMo) or by masking tokens, and predicting the value of the masked tokens using the full remainder of the context.

This task design crucially allows for a multi-layered bidirectional architecture, and consequently a much richer representation of context in each word's pre-trained representation. BERT demonstrates dramatic improvements over prior work when fine tuned on a small amount of supervised data, suggesting that this change added substantial value.

You May Not Need Attention

https://arxiv.org/abs/1810.13409

I admit it - the title of the paper pulled me in, existing as it does in the chain of weirdly insider-meme papers, starting with Vaswani's 2017 "Attention Is All You Need". That paper has been hugely influential, and the domain of machine translation as a whole has begun to move away from processing (or encoding) source sentences with recurrent architectures, to instead processing them using self-attention architectures. (Self-attention is a little too nuanced to go into in full depth here, but the basic idea is: instead of summarizing varying-length sequences by feeding each timestep into a recurrent loop and building up hidden states, generate a query, and weight the contribution of each timestep to each "hidden state" based on the dot product between that query and each timestep's representation). There has been an overall move in recent years away from recurrence being the accepted default for sequence data, and towards attention and (often dilated) convolution taking up more space. I find this an interesting set of developments, and had hopes that this paper would address that arc.

However, unfortunately, the title was quite out of sync with the actual focus of the paper - instead of addressing the contribution of attention mechanisms vs recurrence, or even directly addressing any of the particular ideas posed in the "Attention is All You Need" paper, this YMNNA instead takes aim at a more fundamental structural feature of translation models: the encoder/decoder structure. The basic idea of an encoder/decoder approach, in a translation paradigm, is that you process the entire source sentence before you start generating the tokens of the predicted, other-language target sentence. Initially, this would work by running a RNN over the full sentence, and using the final hidden state of that RNN as a compressed representation of the full sentence. More recently, the norm has been to use multiple layers of RNN, and to represent the source sentence via the hidden states at each timestep (so: as many hidden states as you have input tokens), and then at each step in the decoding process, calculate an attention-weighted average over all of those hidden states. But, fundamentally, both of these structures share the fact that some kind of global representation is calculated and made available to the decoder before it starts predicting words in the output sentence.

This makes sense for a few reasons. First, and most obviously, languages aren't naturally aligned with one another, in the sense of one word in language X corresponding to one word in language Y. It's not possible for you to predict a word in the target sentence if its corresponding source sentence token has not yet been processed. For another, there can be contextual information from the sentence as a whole that can disambiguate between different senses of a word, which may have different translations - think Teddy Bear vs Teddy Roosevelt. However, this paper poses the question: how well can you do if you throw away this structure, and build a model that continually emits tokens of the target sequence as it reads in the source sentence? Using a recurrent model, the YMNNA model takes, at each timestep, the new source token, the previous target token, and the prior hidden state from the last time step of the RNN, and uses that to predict a token.

However, that problem mentioned earlier - of languages not natively being aligned such that you have the necessary information to predict a word by the time you get to its point in the target sequence - hasn't gone away, and is still alive and kicking. This paper solves it in a pretty unsatisfying way - by relying on an external tool, fast-align, that does the work of guessing which source tokens correspond to which target tokens, and inserting buffer tokens into the target, so that you don't need to predict a word until it's already been seen by the source-reading RNN; until then you just predict the buffer. This is fine and clever as a practical heuristic, but it really does make their comparisons against models that do alignment and translation jointly feel a little weak.

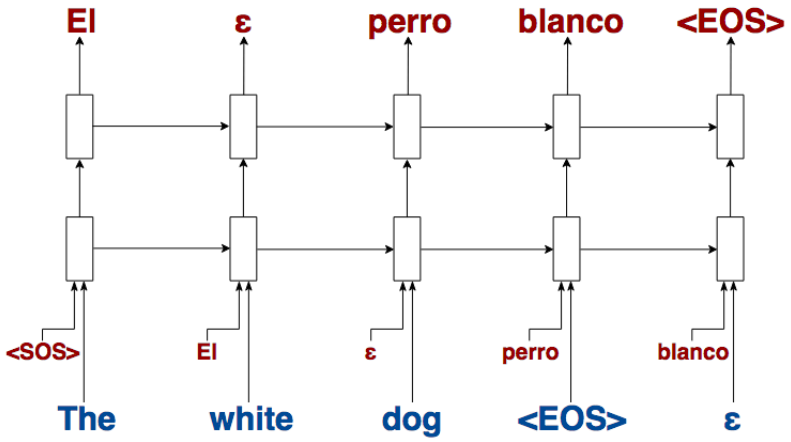


Figure 1: The eager model translating the sentence "The white dog" into Spanish. Source (target) tokens are in blue (red). ε is the padding token, which is removed during postprocessing. The diagram presents an eager translation model with two LSTM layers.

An additional heuristic that makes the overall narrative of the paper less compelling is the fact that, in order to get comparable performance to their baselines, they padded the target sequences with between 3 and 5 buffer tokens, meaning that the models learned that they could process the first 3-5 tokens of the sentence before they need to start emitting the target. Again, there's nothing necessarily wrong with this, but, since they are consuming a portion of the sentence before they start emitting translations, it does make for a less stark comparison with the "read the whole sentence" encoder/decoder framework.

A few other frustrations, and notes from the paper's results section:
As earlier mentioned, the authors don't actually compare their work against the "Attention is All You Need" paper, but instead to a 2014 paper. This is confusing both in terms of using an old baseline for SOTA, and also in terms of their title implicitly arguing they are refuting a paper they didn't compare to

Comparing against their old baseline, their eager translation model performs worse on all sentences less than 60 tokens in length (which makes up the vast majority of all the sentences there are), and only beats the baseline on sentences > 60 tokens in length
Additionally, they note as a sort of throwaway line that their model took almost three times as long to train as the baseline, with the same amount of parameters, simply because it took so much longer to converge.

Being charitable, it seems like there is some argument that an eager translation framework performs well on long sentences, and can do so while only keeping a hidden state in memory, rather than having to keep the hidden states for each source sequence element around, like attention-based decoders require. However, overall, I found this paper to be a frustrating let-down, that used too many heuristics and hacks to be a compelling comparison to prior work.

Trellis Networks

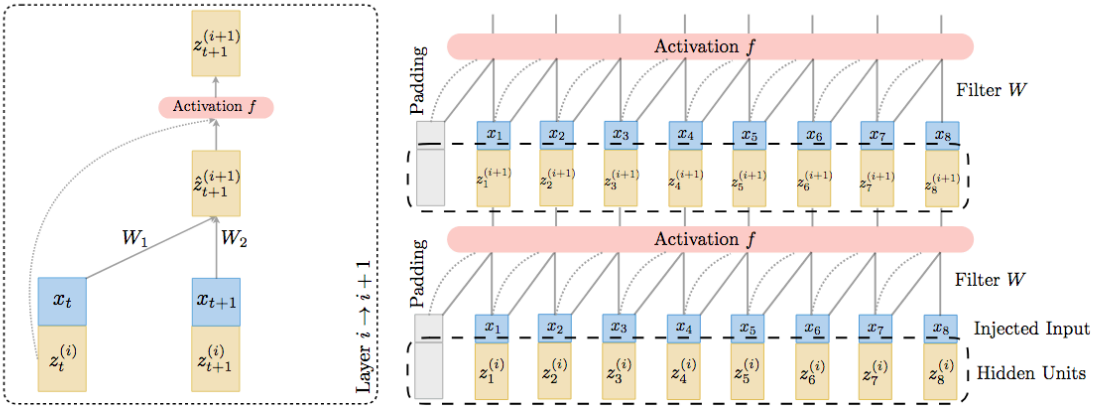
https://arxiv.org/abs/1810.06682

For solving sequence modeling problems, recurrent architectures have been historically the most commonly used solution, but, recently, temporal convolution networks, especially with dilations to help capture longer term dependencies, have gained prominence. RNNs have theoretically much larger capacity to learn long sequences, but also have a lot of difficulty propagating signal forward through long chains of recurrent operations. This paper, which suggests the approach of Trellis Networks, places itself squarely in the middle of the debate between these two paradigms. TrellisNets are designed to be a theoretical bridge between between temporal convolutions and RNNs - more specialized than the former, but more generalized than the latter.

$$\hat{z}_{t+1}^{(i+1)} = W_1 \begin{bmatrix} x_t \\ z_t^{(i)} \end{bmatrix} + W_2 \begin{bmatrix} x_{t+1} \\ z_{t+1}^{(i)} \end{bmatrix}$$

The architecture of TrellisNets is very particular, and, unfortunately, somewhat hard to internalize without squinting at diagrams and equations for awhile. Fundamentally:

- At each layer in a TrellisNet, the network creates a “candidate pre-activation” by combining information from the input and the layer below, for both the current and former time step.
- This candidate pre-activation is then non-linearly combined with the prior layer, prior-timestep hidden state
- This process continues for some desired number of layers.



(a) TrellisNet at an atomic level (b) TrellisNet on a sequence of units

Figure 1: The interlayer transformation of TrellisNet, at an atomic level (time steps t and $t+1$, layers i and $i+1$) and on a longer sequence (time steps 1 to 8, layers i and $i+1$).

At first glance, this structure seems pretty arbitrary: a lot of quantities connected together, but without a clear mechanic for what's happening. However, there are a few things interesting to note here, which will help connect these dots, to view TrellisNet as either a kind of RNN or a kind of CNN:

- TrellisNet uses the same weight matrices to process prior and current timestep inputs/hidden states, no matter which timestep or layer it's on. This is strongly reminiscent of a recurrent architecture, which uses the same calculation loop at each timestep
- TrellisNets also re-insert the model's input at each layer. This also gives it more of a RNN-like structure, where the prior layer's values act as a kind of "hidden state", which are then combined with an input value
- At a given layer, each timestep only needs access to two elements of the prior layer (in addition to the input); it does not require access to all the prior-timestep values of it's own layer. This is important because it means that you can calculate an entire layer's values at once, given the values of the prior layer: this means these models can be more easily parallelized for training

Seeing TrellisNets as a kind of Temporal CNN is fairly straightforward: each timestep's value, at a given layer, is based on a "filter" of the lower-layer value at the current and prior timestep, and this filter is shared across the whole sequence. Framing them as a RNN is certainly trickier, and anyone wanting to understand it in full depth is probably best served by returning to the paper's equations. At at high level, the authors show that TrellisNets can represent a specific kind of RNN: a truncated RNN, where each timestep only uses history from the prior M time steps, rather than the full sequence. This works by sort of imagining the RNN chains as existing along the diagonals of a TrellisNet architecture diagram: as you reach higher levels, you can also reach farther back in time. Specifically, a TrellisNet that wants to represent a depth K truncated RNN, which is allowed to unroll through M steps of history, can do so using M + K - 1 layers. Essentially, by using a fixed operation across layers and timesteps, the TrellisNet authors blur

the line between layer and timestep: any chain of operations, across layers, is fundamentally a series of the same operation, performed many times, and is in that way RNN-like.

The authors have not yet taken a stab at translation, but tested their model on a number of word and character-level language modeling tasks (predicting the next word or character, given prior ones), and were able to successfully beat SOTA on many of them. I'd be curious to see more work broadly in this domain, and also gain a better understanding of areas in which a fixed, recurrently-used layer operation, like the ones used in RNNs and this paper, is valuables, and areas (like a "normal" CNN) where having specific weights for different levels of the hierarchy is valable.

On the Evaluation of Common Sense Reasoning in Natural Language Processing

https://arxiv.org/abs/1811.01778

I should say from the outset: I have a lot of fondness for this paper. It goes upstream of a lot of research-community incentives: It's not methodologically flashy, it's not about beating the State of the Art with a bigger, better model (though, those papers certainly also have their place). The goal of this paper was, instead, to dive into a test set used to evaluate performance of models, and try to understand to what extent it's really providing a rigorous test of what we want out of model behavior. Test sets are the often-invisible foundation upon which ML research is based, but like real-world foundations, if there are weaknesses, the research edifice built on top can suffer.

Specifically, this paper discusses the Winograd Schema, a clever test set used to test what the NLP community calls "common sense reasoning". An example Winograd Schema sentence is:
The delivery truck zoomed by the school bus because it was going so fast.
A model is given this task, and asked to predict which token the underlined "it" refers to. These cases are specifically chosen because of their syntactic ambiguity - nothing structural about the order of the sentence requires "it" to refer to the delivery truck here. However, the underlying meaning of the sentence is only coherent under that parsing. This is what is meant by "common-sense" reasoning: the ability to understand meaning of a sentence in a way deeper than that allowed by simple syntactic parsing and word co-occurrence statistics.

Taking the existing Winograd examples (and, when I said tiny, there are literally 273 of them) the authors of this paper surface some concerns about ways these examples might not be as difficult or representative of "common sense" abilities as we might like.

- First off, there is the basic, previously mentioned fact that there are so few examples that it's possible to perform well simply by random chance, especially over combinatorially large hyperparameter optimization spaces. This isn't so much an indictment of the set itself as it is indicative of the work involved in creating it.

- One of the two distinct problems the paper raises is that of “associativity”. This refers to situations where simple co-occurance counts between the description and the correct entity can lead the model to the correct term, without actually having to parse the sentence. An example here is:
  “I’m sure that my map will show this building; it is very famous.”
  Treasure maps aside, “famous buildings” are much more generally common than “famous maps”, and so being able to associate “it” with a building in this case doesn’t actually require the model to understand what’s going on in this specific sentence. The authors test this by creating a threshold for co-occurrence, and, using that threshold, call about 40% of the examples “associative”
- The second problem is that of predictable structure - the fact that the “hinge” adjective is so often the last word in the sentence, making it possible that the model is brittle, and just attending to that, rather than the sentence as a whole

The authors perform a few tests - examining results on associative vs non-associative examples, and examining results if you switch the ordering (in cases like “Emma did not pass the ball to Janie although she saw that she was open,” where it’s syntactically possible), to ensure the model is not just anchoring on the identity of the correct entity, regardless of its place in the sentence. Overall, they found evidence that some of the state of the art language models perform well on the Winograd Schema as a whole, but do less well (and in some cases even less well than the baselines they otherwise outperform) on these more rigorous examples. Unfortunately, these tests don’t lead us automatically to a better solution - design of examples like this is still tricky and hard to scale - but does provide valuable caution and food for thought.

SeqGAN (GANs for Language)

https://arxiv.org/abs/1609.05473

GANs for images have made impressive progress in recent years, reaching ever-higher levels of subjective realism. However, even as it’s valuable to recognize the areas of GAN’s success, it’s also interesting to think about domains where the GAN architecture is less of a good fit. An example of one such domain is natural language.

As opposed to images, which are made of continuous pixel values, sentences are fundamentally sequences of discrete values: that is, words. In a GAN, when the discriminator makes its assessment of the realness of the image, the gradient for that assessment can be backpropagated through to the pixel level. The discriminator can say “move that pixel just a bit, and this other pixel just a bit, and then I’ll find the image more realistic”. However, there is no smoothly flowing continuous space of words, and, even if you use continuous embeddings of words, it’s still the case that if you tried to apply a small change to a embedding vector, you almost certainly wouldn’t end up with another word, you’d just be somewhere in the middle of nowhere in word space. In short: the discrete nature of language sequences doesn’t allow for gradient flow to propagate backwards through to the generator.

The authors of this paper propose a solution: instead of trying to treat their GAN as one big differentiable system, they framed the problem of "generate a sequence that will seem realistic to the discriminator" as a reinforcement learning problem? After all, this property - of your reward just being generated *somewhere* in the environment, not something analytic, not something you can backprop through - is one of the key constraints of reinforcement learning. Here, the more real the discriminator finds your sequence, the higher the reward. One approach to RL, and the one this paper uses, is that of a policy network, where your parametrized network produces a distribution over actions. You can't update your model to deterministically increase reward, but you can shift around probability in your policy such that your expected reward of following that policy is higher.

This key kernel of an idea - GANs for language, but using a policy network framework to get around not having backprop-able loss/reward- gets you most of the way to understanding what these authors did, but it's still useful to mechanically walk through specifics.

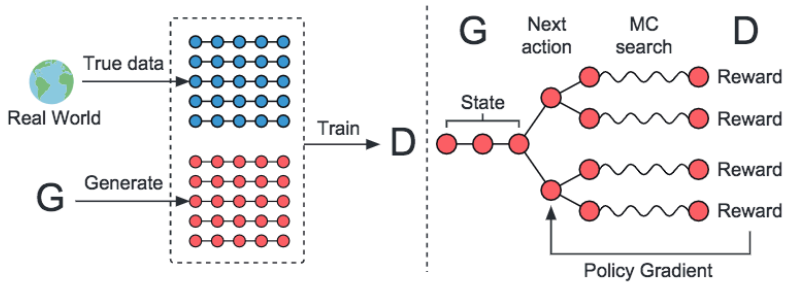


Figure 1: The illustration of SeqGAN. Left: D is trained over the real data and the generated data by G. Right: G is trained by policy gradient where the final reward signal is provided by D and is passed back to the intermediate action value via Monte Carlo search.

- At each step, the "state" was the existing words in the sequence, and the agent's "action" was trying to choose its next word
- The Discriminator could only be applied to completed sequences. So, when the agent was trying to calculate the reward of an action at a state, it would use Monte Carlo Tree Search: randomly "rolling out" many possible futures by randomly sampling from the policy, and then taking the average Discriminator judgment of all those futures resulting from each action as being its expected reward
- The Generator is a LSTM that produces a softmax over words, which can be interpreted as a policy if it's sampled from randomly

Evaluating Language GANs and Neural Text Generation

This paper's high-level goal is to evaluate how well GAN-type structures for generating text are performing, compared to more traditional maximum likelihood methods. In the process, it zooms into the ways that the current set of metrics for comparing text generation fail to give a well-rounded picture of how models are performing.

In the old paradigm, of maximum likelihood estimation, models were both trained and evaluated on a maximizing the likelihood of each word, given the prior words in a sequence. That is, models were good when they assigned high probability to true tokens, conditioned on past tokens. However, GANs work in a fundamentally new framework, in that they aren't trained to increase the likelihood of the next (ground truth) word in a sequence, but to generate a word that will make a discriminator more likely to see the sentence as realistic. Since GANs don't directly model the probability of token t, given prior tokens, you can't evaluate them using this maximum likelihood framework.

This paper surveys a range of prior work that has evaluated GANs and MLE models on two broad categories of metrics, occasionally showing GANs to perform better on one or the other, but not really giving a way to trade off between the two.

- The first type of metric, shorthanded as "quality", measures how aligned the generated text is with some reference corpus of text: to what extent your generated text seems to "come from the same distribution" as the original. BLEU, a heuristic frequently used in translation, and also leveraged here, measures how frequently certain sets of n-grams occur in the reference text, relative to the generated text. N typically goes up to 4, and so in addition to comparing the distributions of single tokens in the reference and generated, BLEU also compares shared bigrams, trigrams, and quadgrams (?) to measure more precise similarity of text.
- The second metric, shorthanded as "diversity" measures how different generated sentences are from one another. If you want to design a model to generate text, you presumably want it to be able to generate a diverse range of text - in probability terms, you want to fully sample from the distribution, rather than just taking the expected or mean value. Linguistically, this would be show up as a generator that just generates the same sentence over and over again. This sentence can be highly representative of the original text, but lacks diversity. One metric used for this is the same kind of BLEU score, but for each generated sentence against a corpus of prior generated sentences, and, here, the goal is for the overlap to be as low as possible

The trouble with these two metrics is that, in their raw state, they're pretty incommensurable, and hard to trade off against one another. The authors of this paper try to address this by observing that all models trade off diversity and quality to some extent, just by modifying the entropy of the conditional token distribution they learn. If a distribution is high entropy, that is, if it spreads probability out onto more tokens, it's likelier to bounce off into a random place, which increases diversity, but also can make the sentence more incoherent. By contrast, if a distribution is too low entropy, only ever putting probability on one or two words, then it will be only ever capable of carving out a small number of distinct paths through word space. The below

table shows a good example of what language generation can look like at high and low levels of entropy

α	
2.0	(1) If you go at watch crucial characters putting awareness in Washington , forget there are now unique developments organized personally then why charge . (2) Front wants zero house blood number places than above spin 5 provide school projects which youth particularly teenager temporary dollars plenty of investors enjoy headed Japan about if federal assets own , at 41 .
1.0	(1) Researchers are expected to comment on where a scheme is sold , but it is no longer this big name at this point . (2) We know you ' re going to build the kind of home you ' re going to be expecting it can give us a better understanding of what ground test we ' re on this year , he explained .
0.7	(1) The other witnesses are believed to have been injured , the police said in a statement , adding that there was no immediate threat to any other witnesses . (2) The company ' s net income fell to 5 . 29 billion , or 2 cents per share , on the same period last year .
0.0	(1) The company ' s shares rose 1 . 5 percent to 1 . 81 percent , the highest since the end of the year . (2) The company ' s shares rose 1 . 5 percent to 1 . 81 percent , the highest since the end of the year .

The entropy of a softmax distribution be modified, without changing the underlying model, by changing the *temperature* of the softmax calculation. So, the authors do this, and, as a result, they can chart out that model's curve on the quality/diversity axis. Conceptually, this is asking "at a range of different quality thresholds, how good is this model's diversity," and vice versa. I mentally analogize this to a ROC curve, where it's not really possible to compare, say, precision of models that use different thresholds, and so you instead need to compare the curve over a range of different thresholds, and compare models on that.

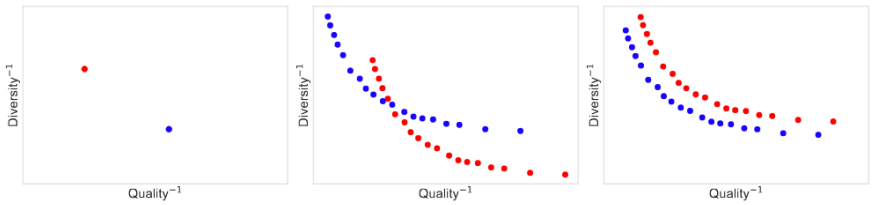


Figure 2: Demonstration of the difficulty of evaluating NLG models on both quality and diversity without temperature. Each sub-figure plots inverse quality against inverse diversity (lower is better for both metrics). **Left:** current way of comparing NLG models. In this case, it is impossible to come to any meaningful conclusions about which model dominates the other. **Middle:** With our proposed NLG evaluation framework, the temperature sweep shines light on the relative performance of the models: the red one should be used for high-diversity samples and the blue one for high-quality. **Right:** A second simulated scenario (consistent with the left Figure) where the temperature sweep reveals that the blue model dominates the red. That is, for any desired diversity-level, there is a temperature for which blue outperforms red in terms of quality (and vice versa).

When they do this for GANs and MLEs, they find that, while GANs might dominate on a single metric at a time, when you modulate the temperature of MLE models, they're able to achieve superior quality when you tune them to commensurate levels of diversity.

Meta Learning / Multi Task

One Shot Imitation Learning

https://arxiv.org/abs/1703.07326

This paper is actually one of the earlier ones in the recent train of meta-learning papers out of UC Berkeley, and even though I've read some of the papers that got built on this, I thought it would be valuable to go back and and read this one as well. The framing of this paper is one of imitation learning, and, specifically, imitation learning using just one example demonstration.

Imitation can be a useful way of learning reinforcement tasks where it's difficult or expensive to program a reward function (you get this many points for your hand closing around the object, this many points for lifting it up, etc), and where it's easier to just show the robot a demonstration of what correct execution looks like, and ask it to mimic the behavior. A standard imitation model would be trained by training a single model that learns to imitate one task, meaning you need different models for different tasks. This paper tries instead to build an architecture that can learn the meta task of "imitate a demonstration, given only one example". To do this, it needs to learn how to extract information from a given demonstration about what we want it to do, and also learn how to translate that compressed description into action. On a very abstracted and conceptual level, I think of this as the difference between a system learning to add 2 + 5, and a system learning to add x + 5, where x will be input at execution time. In the former case, it can earn a hardcoded answer to the operation, while in the latter, it must learn how to perform an operation on arbitrary inputs. (X here is a stand-in for the single demonstration the model must learn to copy from).

The structure of this model works in three main parts:

1. A "demonstration network", which compresses the information contained in the demonstration. Thich can be thought of creating a distilled specification of what task we want the robot to perform at each one-shot instance. The demonstration network first does random downsampling of the demonstration trajectory (which consists of a number of "frames" of the environment, one per time-step). It then performs self-attention, where each block of the environment creates a contextualized representation of itself that pulls in information from all other blocks. Attention works by generating a "query" (which here is done by each block of the environment) and then creating a weighted sum of all the inputs under consideration, where the weight of each input is determined by how well it matches against the query. Since this weight calculation is independent of any positional information of the input, it can be applied to to variable length inputs. The result of this network is is a set of temporal snapshots, where each snapshot is divided

into blocks, and where all the blocks capture both their own information and information about the rest of the snapshot.
2. A "context network," which combines the robot's "current state" information with the aforementioned distillation, to determine what needs to be done now. It does this by performing temporal attention over the different timesteps of the demonstration, to determine which parts are most relevant, given the robot's current state. This results in it having a vector of size num_blocks (since it just took a weighted sum over timesteps, and collapsed that dimension). It then repeats a similar operation, doing attention over the blocks, and from that gets a fixed-length vector that's not constrained by either a fixed number of timesteps or a fixed number of blocks.
3. A "manipulation network", which maps the context network's more conceptual, vector-space specification of a next action into actual physical movements on the part of the robot. This part of the network is less interesting, and is basically just a feed forward network.

Importance Weighted Actor Learner Architectures

https://arxiv.org/abs/1802.01561

This reinforcement learning paper starts with the constraints imposed an engineering problem - the need to scale up learning problems to operate across many GPUs - and ended up, as a result, needing to solve an algorithmic problem along with it.

In order to massively scale up their training to be able to train multiple problem domains in a single model, the authors of this paper implemented a system whereby many "worker" nodes execute trajectories (series of actions, states, and reward) and then send those trajectories back to a "learner" node, that calculates gradients and updates a central policy model. However, because these updates are queued up to be incorporated into the central learner, it can frequently happen that the policy that was used to collect the trajectories is a few steps behind from the policy on the central learner to which its gradients will be applied (since other workers have updated the learner since this worker last got a policy download). This results in a need to modify the policy network model design accordingly.

IMPALA (Importance Weighted Actor Learner Architectures) uses an "Actor Critic" model design, which means you learn both a policy function and a value function. The policy function's job is to choose which actions to take at a given state, by making some higher probability than others. The value function's job is to estimate the reward from a given state onward, if a certain policy p is followed. The value function is used to calculate the "advantage" of each action at a given state, by taking the reward you receive through action a (and reward you expect in the future), and subtracting out the value function for that state, which represents the average future reward you'd get if you just sampled randomly from the policy from that point onward. The policy network is then updated to prioritize actions which are higher-advantage. If you're on-policy, you can calculate a value function without needing to explicitly calculate the probabilities of each action, because, by definition, if you take actions according to your policy

probabilities, then you're sampling each action with a weight proportional to its probability. However, if your actions are calculated off-policy, you need correct for this, typically by calculating an "importance sampling" ratio, that multiplies all actions by a probability under the desired policy divided by the probability under the policy used for sampling. This cancels out the implicit probability under the sampling policy, and leaves you with your actions scaled in proportion to their probability under the policy you're actually updating. IMPALA shares the basic structure of this solution, but with a few additional parameters to dynamically trade off between the bias and variance of the model.

The first parameter, rho, controls how much bias you allow into your model, where bias here comes from your model not being fully corrected to "pretend" that you were sampling from the policy to which gradients are being applied. The trade-off here is that if your policies are far apart, you might downweight its actions so aggressively that you don't get a strong enough signal to learn quickly. However, the policy you learn might be statistically biased. Rho does this by weighting each value function update by

$$\rho_t \overset{\text{def}}{=} \min\left(\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}\right)$$

where rho-bar is a hyperparameter. If rho-bar is high, then we allow stronger weighting effects, whereas if it's low, we put a cap on those weights.

The other parameter is c, and instead of weighting each value function update based on policy drift at that state, it weights each timestep based on how likely or unlikely the action taken at that timestep was under the true policy.

$$\min\left(\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}\right)$$

Timesteps that much likelier under the true policy are upweighted, and, once again, we use a hyperparameter, c-bar, to put a cap on the amount of allowed upweighting. Where the prior parameter controlled how much bias there was in the policy we learn, this parameter helps control the variance - the higher c-bar, the higher the amount of variance there will be in the updates used to train the model, and the longer it'll take to converge.

Multi-Task RL Using Pop Art Normalization

https://arxiv.org/abs/1809.04474

This paper posits that one of the central problems stopping multi-task RL - that is, single models trained to perform multiple tasks well - from reaching better performance, is the inability to balance model resources and capacity between the different tasks the model is being asked to learn. Empirically, prior to this paper, multi-task RL could reach ~50% of human accuracy on Atari and Deepmind Lab tasks. The fact that this is lower than human accuracy is actually somewhat less salient than the fact that it's quite a lot lower than single-task RL - how a single model trained to perform only that task could do.

When learning a RL model across multiple tasks, the reward structures of the different tasks can vary dramatically. Some can have high-magnitude, sparse rewards, some can have low magnitude rewards throughout. If a model learns it can gain what it thinks is legitimately more reward by getting better at a game with an average reward of 2500 than it does with an average reward of 15, it will put more capacity into solving the former task. Even if you apply normalization strategies like reward clipping (which treats all rewards as a binary signal, regardless of magnitude, and just seeks to increase the frequency of rewards), that doesn't deal with some environments having more frequent rewards than others, and thus more total reward when summed over timestep.

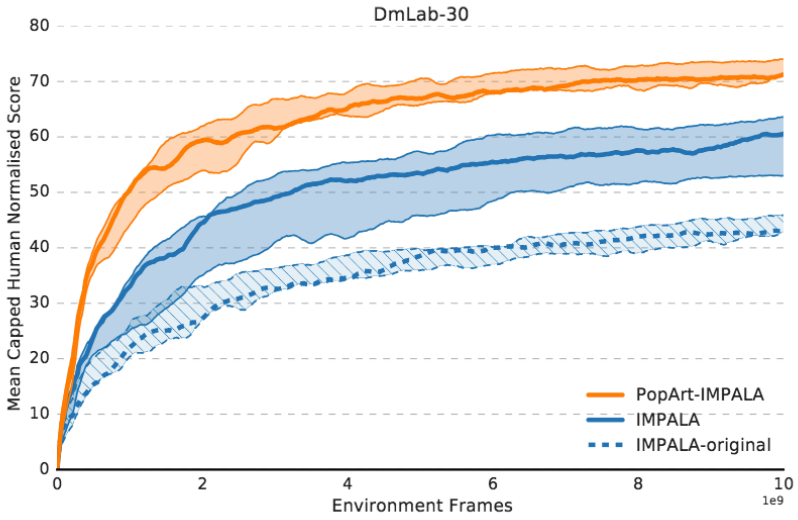
The authors here try to solve this problem by performing a specific kind of normalization, called Pop Art normalization, on the problem. PopArt normalization (don't worry about the name) works by adaptively normalizing both the target and the estimate of the target output by the model, at every step. In the Actor-Critic case that this model is working on, the target and estimate that are being normalized are, respectively, 1) the aggregated rewards of the trajectories from state S onward, and 2) the value estimate at state S. If your value function is perfect, these two things should be equivalent, and so you optimize your value function to be closer to the true rewards under your policy. And, then, you update your policy to increase probability of actions with higher advantage (expected reward with that action, relative to the baseline Value(S) of that state). The "adaptive" part of that refers to correcting for the fact when you're estimating, say, a Value function to predict the total future reward of following a policy at a state, that V(S) will be strongly non-stationary, since by improving your policy you are directly optimizing to increase that value. This is done by calculating "scale" and "shift" parameters off of a recent data.

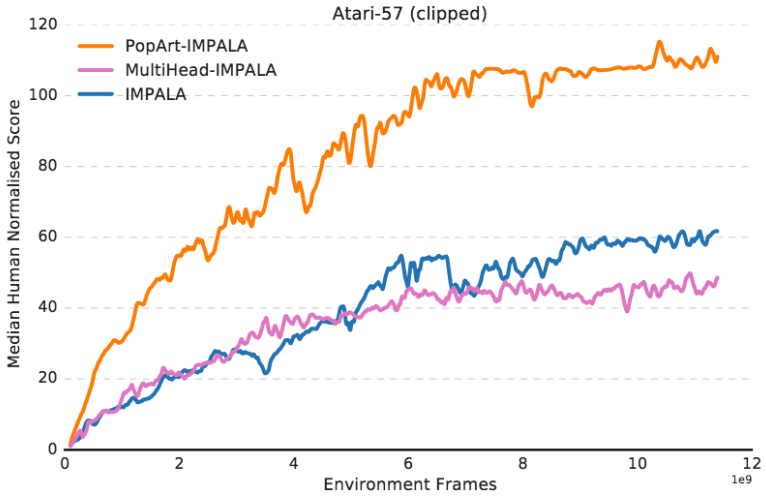
The other part of the PopArt algorithm works by actually updating the estimate our model is producing, to stay normalized alongside the continually-being-re-normalized target.

$$\mathbf{w}'_i = \frac{\sigma_i}{\sigma'_i}\mathbf{w}_i\,, \qquad b'_i = \frac{\sigma_i b_i + \mu_i - \mu'_i}{\sigma'_i}\,,$$

It does this by taking the new and old versions of scale (sigma) and shift (mu) parameters (which will be used to normalize the target) and updates the weights and biases of the last layer, such that the movement of the estimator moves along with the movement in the target.

Using this toolkit, this paper proposes learning one *policy* that's shared over all task, but keeping shared value estimation functions for each task. Then, it normalizes each task's values independently, meaning that each task ends up contributing equal weight to the gradient updates of the model (both for the Value and Policy updates). In doing this, the authors find dramatically improved performance at both Atari and Deepmind, relative to prior IMPALA work





Proximal Meta Policy Search

https://arxiv.org/abs/1810.06784

This paper dives into two kinds of meta-learning, and tries to build up a theoretical foundation for what the full analytic form of their gradient looks like. More specifically, the paper illustrates a difference in how the two kinds of meta-learning treat the dependence of the task-specific performance on the parameters of the model prior to task-specific adaptation.

To refresh on how meta-learning works: the idea is that you "learn to learn" by training a model to able to quickly learn a new task. One approach to this is to use a "vanilla" learning method (like, in reinforcement learning, a policy gradient method) and simply wrap it in a meta-learning loop, where you consider as each "observation" an instance of a task that is sampled, and where a small number of gradient update steps are performed. You then update the initial parameters of the model to get the highest expected reward over all of these "quick learn on new task" observations. The other approach is to train a recurrent model with complicated internal updating dynamics, and for each instance of a task feed all of the observations into the model sequentially, updating the "quick weights" (i.e. RNN hidden states), which are considered analogous to the parameters of a model post-update on a specific task. When the RNN state has been updated by taking in all the observations of a certain task, it should then be "trained" on that task, and able to produce correct outputs for test observations.

This paper dives into two different formulations of the former, gradient-based learning objective, which it calls Formulation I and Formulation II. I initially misunderstood this, and thought the two formulations referred to the RNN vs Gradient-based methods, but upon re-read, Formulation II does indeed refer to another version of the gradient-based method, E-MAML.

The main argument of this paper is that, even though these formulations are ostensibly optimizing the same loss function in expectation, one treats the task-specific update step as stochastic, and the other treats it as deterministic, and that this has implications on their performance. I'll be honest and say that the mathematical and theoretical steps that go between the theoretical formulation of the two approaches, and the analysis of their gradients, wasn't something I absorbed on this run-through. However, I think I can elucidate a bit of the paper's conclusions, even if I can't independently vouch for it's logic.

When you're updating theta parameters for a meta-learning model, you can divide the gradient calculation into two parts: one which optimizes your pre-task-update parameters theta to produce better post-update trajectories (in terms of having higher reward), and one which optimizes those same pre-task-update policy parameters to produce better pre-update trajectories. "Better" pre-update trajectories here means: pre-update trajectories that lead to more useful gradient updates. This is important in terms of reinforcement learning because our initialization parameters don't only impact where we "start from" (where you could imagine that meta learning incentivizes parameters to start in a sort of "central" place between tasks), but

also impact the trajectory we'll sample on the new task, which impacts the gradient updates we make between our initial parameters and our new ones.

$$\nabla_\theta J^{II}_{\text{pre}}(\boldsymbol{\tau}, \boldsymbol{\tau}') = \alpha \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}')$$

$$\nabla_\theta J^{I}_{\text{pre}}(\boldsymbol{\tau}, \boldsymbol{\tau}') = \alpha \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) \Bigg(\underbrace{(\nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}))^\top}_{\nabla_\theta J^{\text{inner}}} \underbrace{(\nabla_{\theta'} \log \pi_{\theta'}(\boldsymbol{\tau}') R(\boldsymbol{\tau}'))}_{\nabla_{\theta'} J^{\text{outer}}} \Bigg)$$

The Formulation II approach treats the whole problem as one big reinforcement learning environment: you start with some initial parameters theta, that results in some post-update reward, and everything in between is a black box that you'd optimize the same way you'd optimize an environmental reward function whose dynamics are unknown. By contrast, Formulation I explicitly writes out a causal chain by which the initial parameters theta impact the trajectories sampled on a new task. Because of the inner product, shown above, we're optimizing more directly for initial policies that will lead to large (informative) update steps, which has the impact of making it easier to learn models that adapt well, relative to the Formulation II approach of treating the dynamics as a fully black box system to be sampled from.

There was more to this paper, focusing on the difficulties of calculating Hessians of Reinforcement Learning surrogate objective functions, but I don't think I understood enough of it to be worth discussing here.

Intrinsic Rewards/Model-Based RL

Curiosity Driven Learning

https://arxiv.org/abs/1808.04355

I really enjoyed this paper - in addition to being a clean, fundamentally empirical work, it was also clearly written, and had some pretty delightful moments of quotable zen, which I'll reference at the end. The paper's goal is to figure out how far curiosity-driven learning alone can take reinforcement learning systems, without the presence of an external reward signal. "Intrinsic" reward learning is when you construct a reward out of internal, inherent features of the environment, rather than using an explicit reward function. In some ways, intrinsic learning in RL can be thought of as analogous to unsupervised learning in classification problems, since reward functions are not inherent to most useful environments, and (when outside of game environments that inherently provide rewards), frequently need to be hand-designed. Curiosity-driven learning is a subset of intrinsic learning, which uses as a reward signal the difference between a prediction made by the dynamics model (predicting next state, given action) and the true observed next state. Situations where the this prediction area are high generate high reward for the agent, which incentivizes it to reach those states, which allows the dynamics model to then make ever-better predictions about them.

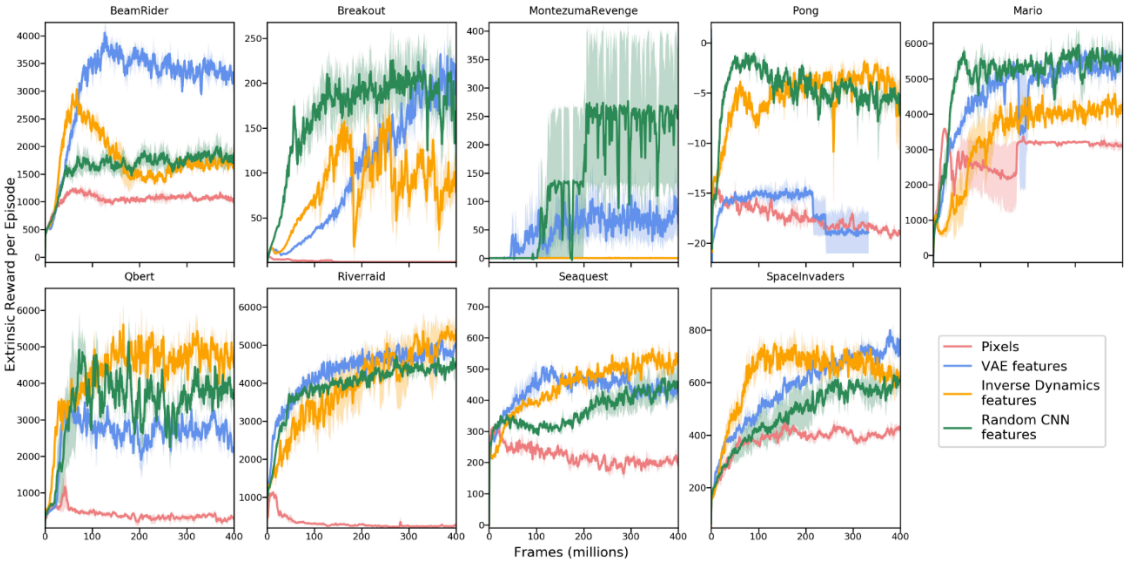
Two key questions this paper raises are:

1) Does this approach even work when used on it's own? Curiosity had previously most often been used as a supplement to extrinsic rewards, and the authors wanted to know how far it could go separately.

2) What is the best feature to do this "surprisal difference" calculation in? Predicting raw pixels is a high-dimensional and noisy process, so naively we might want something with fewer, more informationally-dense dimensions, but it's not obvious which methods that satisfy these criteria will work the best, so the paper empirically tried them.

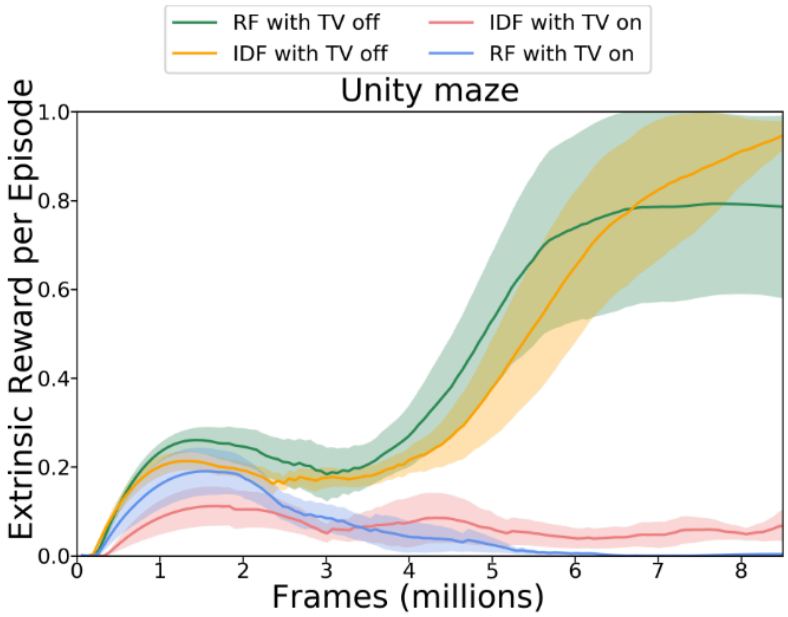
The answer to (1) seems to be: yes, at least in the video games tested. Impressively, when you track against extrinsic reward (which, again, these games have, but we're just ignoring in a curiosity-only setting), the agents manage to increase it despite not optimizing against it directly. There were some Atari games where this effect was stronger than others, but overall performance was stronger than might have been naively expected. One note the authors made, worth keeping in mind, is that it's unclear how much of this is an artifact of the constraints and

incentives surrounding game design, which might reflect back a preference for gradually-increasing novelty because humans find it pleasant.



As for (2), another interesting result of this paper is that random features performed consistently well as a feature space to do these prediction/reality comparisons in. Random features here is really just as simple as “design a convolutional net that compresses down to some dimension, randomly initialize it, and then use those randomly initialized weights to run forward passes of the network to get your lower-dimensional state”. This has the strong disadvantage of (presumably) not capturing any meaningful information about the state, but also has the advantage of being stable: the other techniques tried, like pulling out the center of a VAE bottleneck, changed over time as they were being trained on new states, so they were informative, but non-stationary.

My two favorite quotable moments from this paper were:
when the authors noted that they had removed the “done” signal associated with an agent “dying,” because it is itself a sort of intrinsic reward. However, “in practice, we do find that the agent avoids dying in the games since that brings it back to the beginning of the game, an area it has already seen many times and where it can predict the dynamics well.”. Short and sweet: “Avoiding death, because it’s really boring”



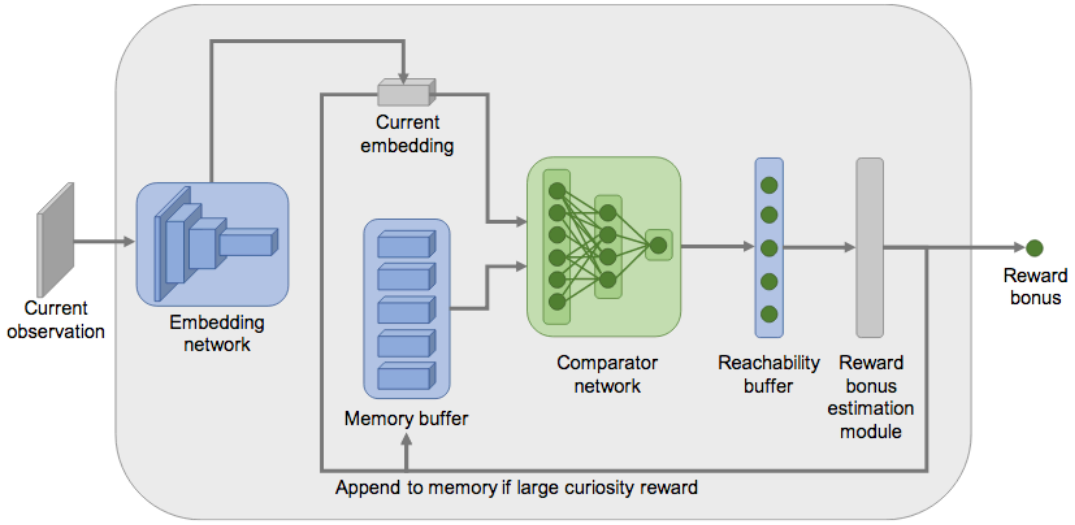
When they noted that an easy way to hack the motivation structure of a curiosity-driven agent was through a "noisy tv", which, every time you pressed the button, jumped to a random channel. As expected, when they put this distraction inside a maze, the agent spent more time jacking up reward through that avenue, rather than exploring. Any resemblance to one's Facebook feed is entirely coincidental.

Episodic Curiosity Through Reachability

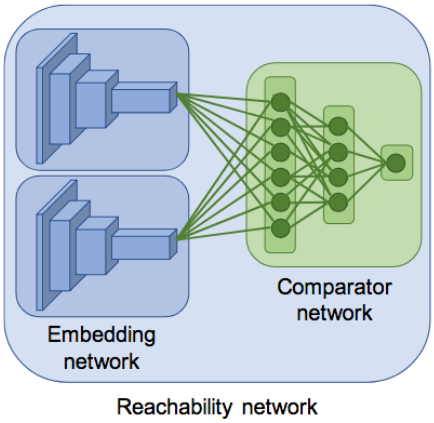
https://arxiv.org/abs/1810.02274

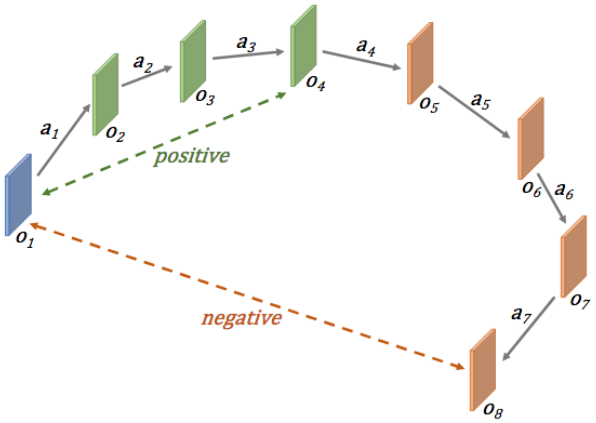
This paper proposes a new curiosity-based intrinsic reward technique that seeks to address one of the failure modes of previous curiosity methods. The basic idea of curiosity is that, often, exploring novel areas of an environment can be correlated with gaining reward within that environment, and that we can find ways to incentivize the former that don't require a hand-designed reward function. This is appealing because many useful-to-learn environments either lack inherent reward altogether, or have reward that is very sparse (i.e. no signal until you reach the end, at which point you get a reward of 1). In both of these cases, supplementing with some kind of intrinsic incentive towards exploration might improve performance. The existing baseline curiosity technique is called ICM, and works based on "surprisal": asking the agent to predict the next state as a function of its current state, and incentivizing exploration of areas where the gap between these two quantities is high, to promote exploration of harder-to-predict (and presumably more poorly sampled) locations. However, one failure mode of this approach is something called the "noisy TV" problem, whereby if the environment contains something analogous to a television where one can press a button and go to a random channel, that is highly unpredictable, and thus a source of easy rewards, and thus liable to distract the agent from any other actions.

As an alternative, the authors here suggest a different way of defining novelty: rather than something that is unpredictable, novelty should be seen as something far away from what I as an agent have seen before. This is more direct than the prior approach, which takes 'hard to predict' as a proxy for 'somewhere I haven't explored', which may not necessary be a reasonable assumption.



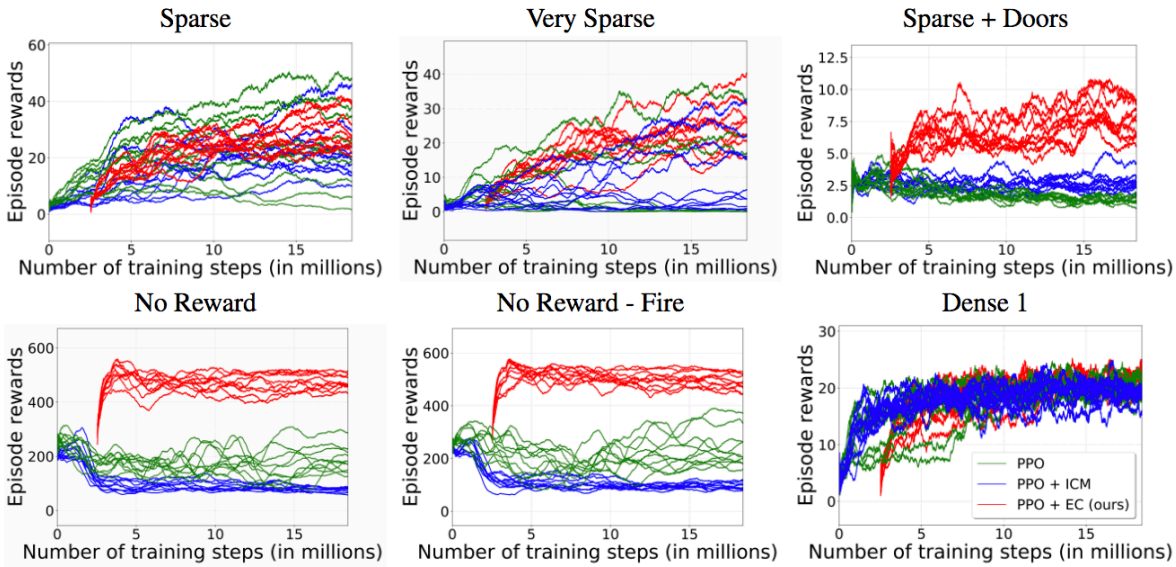
They implement this idea by keeping a memory of past (embedded) observations that the agent has seen during this episode, and, at each step, check whether the current observation is predicted to be more than K steps away than any of the observations in memory (more on that in a moment). If so, a bonus reward is added, and this observation is added to the aforementioned memory. (Which, waving hands vigorously, kind of ends up functioning as a spanning set of prior experience).





The question of "how many steps is observation A from observation B" is answered by a separate Comparator network which is trained in pretty straightforward fashion: a random-samplling policy is used to collect trajectories, which are then turned into pairs of observations as input, and a 1 if they occurred > k + p steps apart, and a 0 if they occurred < k steps apart. Then, these paired states are passed into a shared-weight convolutional network, which creates an embedding, and, from that embedding, a prediction is made as to whether they're closer than the thresholds or farther away. This network is pre-trained before the actual RL training starts. (Minor sidenote: at RL-training time, the network is chopped into two, and the embedding read out and stored, and then input as a pair with each current observation to make the prediction).

Overall, the authors find that their method works better than both ICM and no-intrinsic-reward for VizDoom (a maze + shooting game), and the advantage is stronger in situations more sparse settings of the external reward.



On DeepMind Lab tasks, they saw no advantage on tasks with already-dense extrinsic rewards, and little advantage on the "normally sparse", which they suggest may be due to it actually being easier than expected. They added doors to a maze navigation task, to ensure the agent couldn't

find the target right away, and this situation brought better performance of their method. They also tried a fully no-extrinsic-reward situation, and their method strongly performed both the ICM baseline and (obviously) the only-extrinsic-reward baseline, which was basically an untrained random policy in this setting. Regarding the poor performance of the ICM baseline in this environment, "we hypothesise that the agent can most significantly change its current view when it is close to the wall — thus increasing one-step prediction error — so it tends to get stucknear "interesting" diverse textures on the walls.".

Model-Based Active Exploration

https://arxiv.org/abs/1810.12162

This paper continues in the tradition of curiosity-based models, which try to reward models for exploring novel parts of their environment, in the hopes this can intrinsically motivate learning. However, this paper argues that it's insufficient to just treat novelty as an occasional bonus on top of a normal reward function, and that instead you should figure out a process that's more specifically designed to increase novelty. Specifically: you should design a policy whose goal is to experience transitions and world-states that are high novelty.

In this setup, like in other curiosity-based papers, "high novelty" is defined in terms of a state being unpredictable given a prior state, history, and action. However, where other papers saw novelty reward as something only applied when the agent arrived at somewhere novel, here, the authors build a model (technically, an ensemble of models) to predict the state at various future points. The ensemble is important here because it's (quasi) bootstrapped, and thus gives us a measure of uncertainty. States where the predictions of the ensemble diverge represent places of uncertainty, and thus of high value to explore. I don't actually fully follow the analytic specification of this idea (even though the heuristic/algorithmic description makes sense). The authors frame the Utility function of a state and action as being equivalent to the Jenson Shannon Divergence (~distance between probability distributions) shown below.

$$\mathcal{U}(s,a) = \mathrm{JSD}\{\mathcal{P}(\mathcal{S}|s,a,\bar{\mathcal{T}}) \mid \bar{\mathcal{T}} \sim \mathcal{P}(\Gamma)\}$$

Here, P(S | S, a, T) is the probability of a state given prior state and action under a given model of the environment (Transition Model), and P(gamma) is the distribution over the space of possible transition models one might learn. A "model" here is one network out of the ensemble of networks that makes up our bootstrapped (trained on different sets) distribution over models. Conceptually, I think this calculation is measuring "how different is each sampled model/state distribution from all the other models in the distribution over possible models". If the models within the distribution diverge from one another, that indicates a location of higher uncertainty.

What's important about this is that, by building a full transition model, the authors can calculate the expected novelty or "utility" of future transitions it might take, because it can make a best guess based on this transition model (which, while called a "prior", is really something trained on all data up to this current iteration). My understanding is that these kinds of models function similarly to a Q(s,a) or V(s) in a pure-reward case: they estimate the "utility reward" of different states and actions, and then the policy is updated to increase that expected reward.

I've recently read papers on ICM, and I was a little disappointed that this paper didn't appear to benchmark against that, but against Bootstrapped DQN and Exploration Bonus DQN, which I know less well and can less speak to the conceptual differences from this approach. Another difficulty in actually getting a good sense of results was that the task being tested on is fairly specific, and different from RL results coming out of the world of e.g. Atari and Deep Mind Labs. All of that said, this is a cautiously interesting idea, if the results generate to beat more baselines on more environments.

Combined Reinforcement Learning via Abstract Representations

https://arxiv.org/abs/1809.04506

This paper was interesting, despite there being aspects of it I realize I don't fully understand yet. It starts by setting up a contrast between model-based and model-free reinforcement learning, and then discusses how its proposed method, CRAR, combines these two structures, in a way that it argues has valuable side benefits. I found this distinction weird and difficult to initially understand, because the word "model" is quite overloaded, and has evolved to reasonably refer to all kinds of machine learning architectures that are technically model-free, in the sense made here.

Model-based learning reinforcement learning is an older and more traditional style of RL, and has its roots in strategic planning within a known environment. Under this framework, to operate effectively, you needed to plan, and in order to plan, you had to have an understanding of how the environment worked: what consequences and rewards would result from which actions. So, in an environment where the transition dynamics are unknown rather than given, a machine learning model's job is to learn them. Once you have a model that can predict the next states that result from actions, that can be used to plan, because you can simulated what would happen if you took series' of actions in the "real" environment.

Model-free machine learning doesn't try to directly or explicitly predict next states. Instead, it just predicts the expected reward that can come from a given state, or from a given action at a given state. This estimate obviously implicitly contains information about which states will be visited in future, but doesn't actually provide a model of the environment, in the sense of something that could be sampled and simulated from.

The authors here propose an architecture with both model-free and model-based components, which both use a shared embedding/lower-dimensional representation of the input observations ("states"). The approach, shown below, works by training a Q Learning model, and also an environment model, and having both sit atop a shared representation. When choosing an action, these two components work together: at a given state, Q values are calculated for all actions. For the top beta (hyperparameter) actions, the model simulates taking the action by predicting what the next state would be. Then, the process repeats, finding the top beta actions at this new state. This continues until some predetermined depth, and then the trajectories resulting from this "sampling" are examined for which is higher reward. An important note here is that the environment model is predicting the next state *in abstract/low-dimensional space*, not in pixels, which is much easier to do.

The paper makes the claim that it can produce usefully interpretable low-dimensional representations of states using this method, and that it can contribute to more broadly interpretable AI. While this seems true to me of their method, I don't quite understand why it wouldn't be equally true of a model-free network, that pulled down to a similar low-dimensional space, but without a model-based component.

Agent Empowerment

https://uhra.herts.ac.uk/bitstream/handle/2299/1114/901241.pdf?sequence=1

Today I went down the rabbit hole of a (delightful) old paper: I started out wanting to read a more recent paper on intrinsic reward functions, and found that it referenced this older, more foundational paper.

This paper (which I highly recommend even for the less technical, thanks to how clearly it's written), discusses the idea of "empowerment", as a proposed utility function for agents operating without any explicit short-term reward function offered to them by the universe. After all, as the paper wryly notes, "evolutionary feedback via death seems to be hardly sufficient". The goal of this paper is not so much to design a reward function one can use to train an AI, and more to hypothesize about what underlying reward functions might be driving animals in the real world, and theorize a bit about what behaviors one might expect if this were actually what was being implemented "under the hood".

The idea of empowerment is that, all else equal, an agent should prioritize having a stronger ability to impact the world. The different possible states of the world the agent can reach, the more meaningfully different actions it can perform, the better it's likely to be able to adapt to unexpected adverse conditions. The intuition behind this is that the more empowered an agent is to change it's environment, the more able it will be to change it to its own advantage.

The authors measure this notion of empowerment by quantifying the answer to the question: “what is the maximum amount of impact the agent is able to have on its environment through its actions”. Taken down to one higher level of specificity: “how much statistical information from its actions is the agent able to push into being expressed in the environment”. If states carry statistical information from actions, that means that there was more likely a causal relationship between the action and those states being a certain way, which quantifies our desired notion of “impact on the world”. Specifically, the authors frame “agent’s actions” as a sending signal, and “world state several steps ahead” as a receiving signal, and model the “channel capacity” of that channel, which comes down to the maximum mutual information between the sending and receiving channel, across all possible actions (“messages”) that might be taken.

One note here is that the above was slightly incorrect: the frame of empowerment doesn’t actually measure the agent’s actions’ impact on the *world state*, but rather the agent’s actions’ impact on its perception of the world state, through it’s sensors. This leads to the interesting conclusion that if this is truly an internal objective function animals and other agents are optimizing, one strategy might be to optimize one’s sensors as well as one’s actions, to give an overall sense of empowerment. This is a reasonable constraint because it seems unrealistic that a given agent would have, in the paper’s words, a “god’s eye view” of the environment, and it would in fact be limited to information coming from its own sensors.

Recurrent World Models Facilitate Policy Evolution

https://arxiv.org/abs/1809.01999

This paper, coauthored by Schmidhuber and Ha, constructs a reinforcement learning with two key novel properties: 1) Most of the models complexity is contained within parts of the network not trained on reward, and 2) consequently, the actual number of parameters used to choose an action are small enough that they can be learnt through evolution, rather than gradient descent. These properties are interesting because they mimic many of the conditions of a real organism’s environment: that much of the world doesn't come with a direct reward function, and the organism mostly needs to learn to predict the world, and only occasionally learn how to maximize reward using that model of the world. This is accomplished by creating a modular network in three parts:

- A state encoder, V, to compress information about each environment frame into a compact, low-dimensional representation
- A recurrent world model, W, that takes in this compressed state, and is trained to predict the next compressed state
- And, finally, a controller, C, that takes in the current encoded state and the hidden state of the world model (which is presumed to contain information about the predicted future).

An important thing to note here is that the first two models don't require reward at all, they just need to observe the states and state changes of the environment, which can be done under a totally random policy, or by using historical data without a reward. The part of the network that requires reward to train is a single layer neural net, mapping from the two compressed states that are C's inputs to its action probabilities. Because this is so simple it's parameters can actually be evolved rather than learned.

One interesting consequence of this design is that you end up with a predictive model that can simulate transitions in the outside world, because to doesn't need to do its prediction in high dimensional pixel space. Also, given that action probabilities are calculated with a softmax, the authors run a test where they increase the temperature of the softmax in order to push up low action probabilities, to make low probability actions more likely, and thus the environment as a whole "harder" or more unpredictable. Using this, they use this higher-uncertainty world as a kind of high difficulty training regime, with the hopes that it it will then transfer to the "easier" real world game, and, as a whole, they find this to be the case.

(A general comment I have about RL papers, that this paper brought to mind: because so many of them seem to focus on different tasks, it can be hard to compare them directly. So, with that context, I don't feel like I have a good sense of how well this method performs directly compared to more recent model-free ML, but it is if nothing else a well-explained proof of concept for the model-based approach)

Learning Plannable Representations with Causal InfoGAN

https://arxiv.org/abs/1807.09341

This paper tries to solve the problem of how to learn systems that, given a starting state and a desired target, can earn the set of actions necessary to reach that target. The strong version of this problem requires a planning algorithm to learn a full set of actions to take the agent from state A to B. However, this is a difficult and complex task, and so this paper tries to address a relaxed version of this task: generating a set of "waypoint" observations between A and B, such that each successive observation is relatively close to one another in terms of possible actions (the paper calls this 'h-reachable', if observations are reachable from one another in h timesteps). With these checkpoint observations in hand, the planning system can them solve many iterations of a much shorter-time-scale version of the problem.

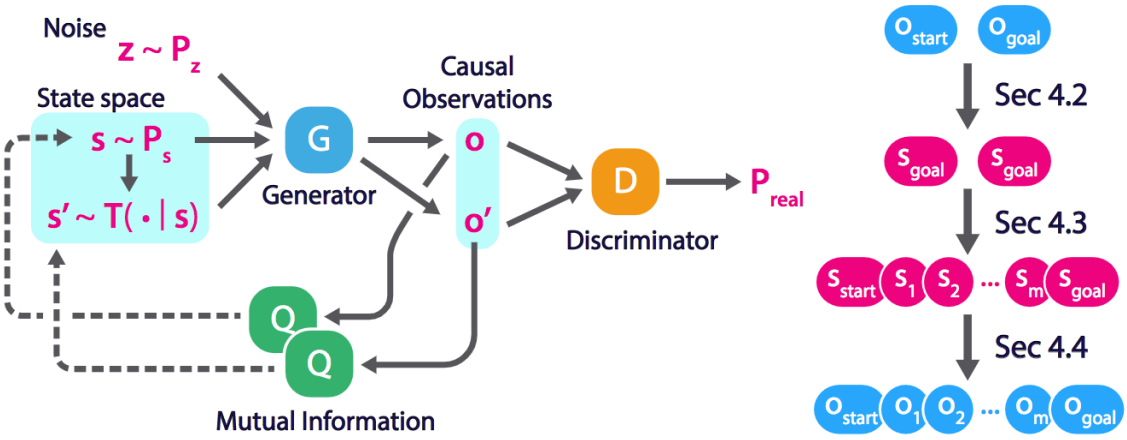
However, the paper asserts, applying pre-designed planning algorithms in observation space (sparse, high-dimensional) is difficult, because planning algorithms apparently do better with denser representations. (I don't really understand, based on just reading this paper, *why* this is the case, other than the general fact that high dimensional, sparse data is just hard for most things). Historically, a typical workflow for applying planning algorithms to an environment would have been to hand-design feature representations where nearby representations were

close in causal decision space (i.e. could be easily reached from one another). This paper's goal is to derive such representations from data, rather than hand-designing them.

The system they design to do this is a little unwieldy to follow, and I only have about 80% confidence that I fully understand all the mechanisms. One basic way you might compress high-dimensional space into a low-dimensional code is by training a Variational Autoencoder, and pulling the latent code out of the bottleneck in the middle. However, we also want to be able to map between our low-dimensional code and a realistic observation space, once we're done planning and have our trajectory of codes, and VAE typically have difficulty generating high-dimensional observations with high fidelity. If what you want is image-generation fidelity, the natural step would be to use a GAN.

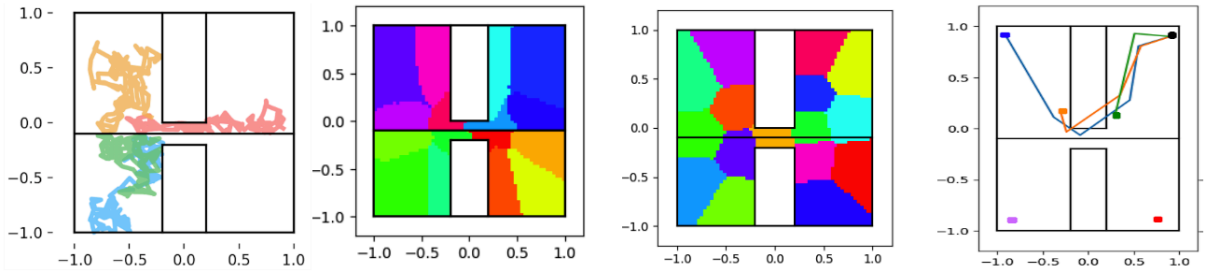
However, GANs aren't really natively designed to learn an informative representation; their main goal is generation, and there's no real incentive for the noise variables used to seed generation to encode any useful information. One GAN design that tries to get around this is the InfoGAN, which gets its name from the requirement that there be high mutual information between (some subset of) the noise variables used to seed the generator, and the actual observation produced. I'm not going to get into the math of the variational approximation, but what this actually mechanically shakes out to is: in addition to generating an observation from a code, an InfoGAN also tries to predict the original code subset given the observation. Intuitively, this requirement, for the observation to contain information about the code, also means the code is forced to contain meaningful information about the image generated from it.

However, even with this system, even if each code separately corresponds to a realistic observation, there's no guarantee that closeness in state space corresponds to closeness in "causality space". This feature is valuable for planning, because it means that if you chart out a trajectory through state space, it actually corresponds to a reasonable trajectory through observation space. In order to solve this problem, the authors added their final, and more novel, modification to the InfoGAN framework: instead of giving the GAN one latent code, and having it predict one observation, they would give two at a time, and have the GAN try to generate a pair of temporally nearby (i.e. less than h actions away) observations. Importantly, they'd also define some transition or sampling function within state space, so that there would be a structured or predictable way that adjacent pairs of states looked. So, if the GAN were able to learn to map adjacent points in state space to adjacent points in observation space, then you'd be able to plan out trajectories in state space, and have them be realistic in observation space.



(a) Causal InfoGAN model (b) Planning paradigm

They do some experiments and do show that both adding the "Info" structure of the InfoGAN, and adding the paired causal structure, lead to states with improved planning properties.They also compared the clusters derived from their Causal InfoGAN states to the clusters you'd get from just naively assuming that nearness in observation space meant nearness in causality space.



(a) (b) (c) (d)

They specifically tested this on an environment divided into two "rooms", where there were many places where there were two points, nearby in Euclidean space, but far away (or mutually inaccessible) in action space. They showed that the Causal InfoGAN was successfully able to learn representations such that points nearby in action space clustered together.

Counterfactually Guided Policy Search

https://arxiv.org/abs/1811.06272

It is a fact universally acknowledged that a reinforcement learning algorithm not in possession of a model must be in want of more data. Because they generally are. Joking aside, it is broadly understood that model-free RL takes a lot of data to train, and, even when you can design them

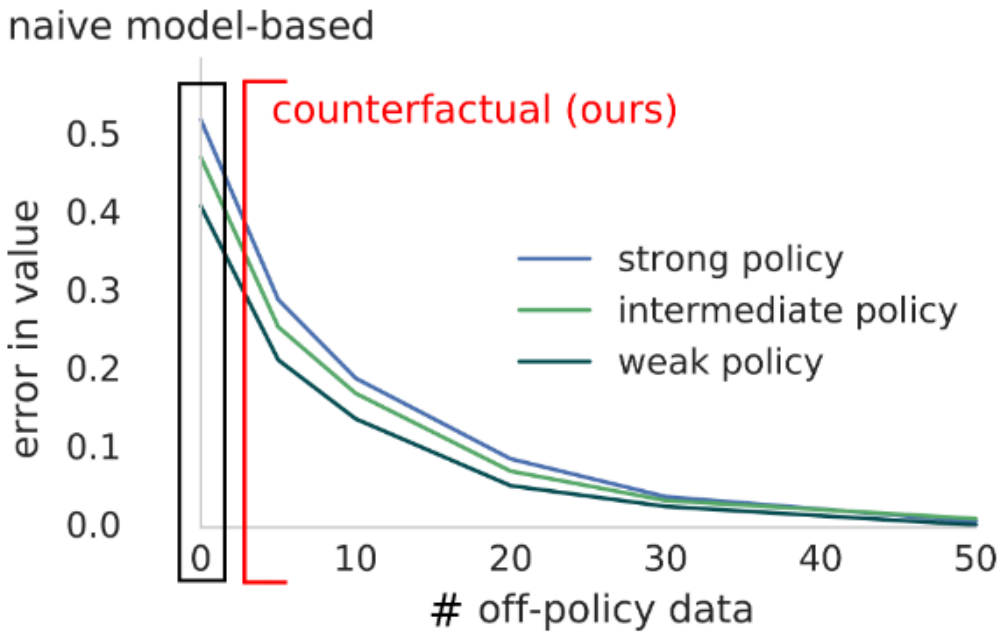
to use off-policy trajectories, collecting data in the real environment might still be too costly. Under those conditions, we might want to learn a model of the environment and generate synthesized trajectories, and train on those. This has the advantage of not needing us to run the actual environment, but the obvious disadvantage that any model will be a simplification of the true environment, and potentially an inaccurate one. These authors seek to answer the question of: "is there a way to generate trajectories that has higher fidelity to the true environment." As you might infer from the fact that they published a paper, and that I'm now writing about it, they argue that, yes, there is, and it's through explicit causal/counterfactual modeling.

Causal modeling is one of those areas of statistics that seems straightforward at its highest level of abstraction, but tends to get mathematically messy and unintuitive when you dive into the math. So, rather than starting with equations, I'm going to try to verbally give some intuitions for the way causal modeling is framed here. Imagine you're trying to understand what would happen if a person had gone to college. There's some set of information you know about them, and some set of information you don't, that's just random true facts about them and about the universe. If, in the real world, they did go to college, and you want to simulate what would have happened if they didn't, it's not enough to just know the observed facts about them, you want to actually isolate all of the random other facts (about them, about the world) that weren't specifically "the choice to go to college", and condition on those as well. Obviously, in the example given here, it isn't really practically possible to isolate all the specific unseen factors that influence someone's outcome. But, conceptually, this quantity, is what we're going to focus on in this paper.

Now, imagine a situation where a RL agent has been dropped into a maze-like puzzle. It has some set of dynamics, not immediately visible to the player, that make it difficult, but ultimately solvable. The best kind of simulated data, the paper argues, would be to keep that state of the world (which is partially unobservable) fixed, and sample different sets of actions the agent might take in that space. Thus, "counterfactual modeling": for a given configuration of random states in the world, sampling different actions within it. To do this, you first have to infer the random state the agent is experiencing. In the normal model-based case, you'd have some prior over world states, and just sample from it. However, if you use the experience of the agent's trajectory, you can make a better guess as to what world configuration it was dropped into. If you can do this, which is, technically speaking, sampling from the posterior over unseen context, conditional on an agent's experience, then the paper suggests you'll be able to generate data that's more realistic, because the trajectories will be direct counterfactuals of "real world" scenarios, rather than potentially-unsolvable or unrealistic draws from the prior.

This is, essentially, the approach proposed by the paper: during training, they make this "world state" visible to the agent, and let it learn a model predicting what state it started with, given some trajectory of experience. They also learn a model that predicts the outcome and ultimately the value of actions taken, conditioned on this random context (as well as visible context, and the agent's prior actions). They start out by using this as a tool for policy evaluation, which is a nice problem setup because you can actually check how well you're doing against some baseline: if you want to know how good your simulated data is at replicating the policy reward on real data, you can just try it out on real data and see. The authors find that they reduce policy reward

estimation error pretty substantially by adding steps of experience (in Bayesian terms, bit of evidence moving them from the prior, towards the posterior).



They also experiment with using this for actual policy search, but, honestly, I didn't quite follow the intuitions behind Guided Policy Search, so I'm just going to not dive into that for now, since I think a lot of the key contributions of the paper are wrapped up in the idea of "estimate the reward of a policy by simulating data from a counterfactual trajectory"

The Impact of Entropy on Policy Regularization

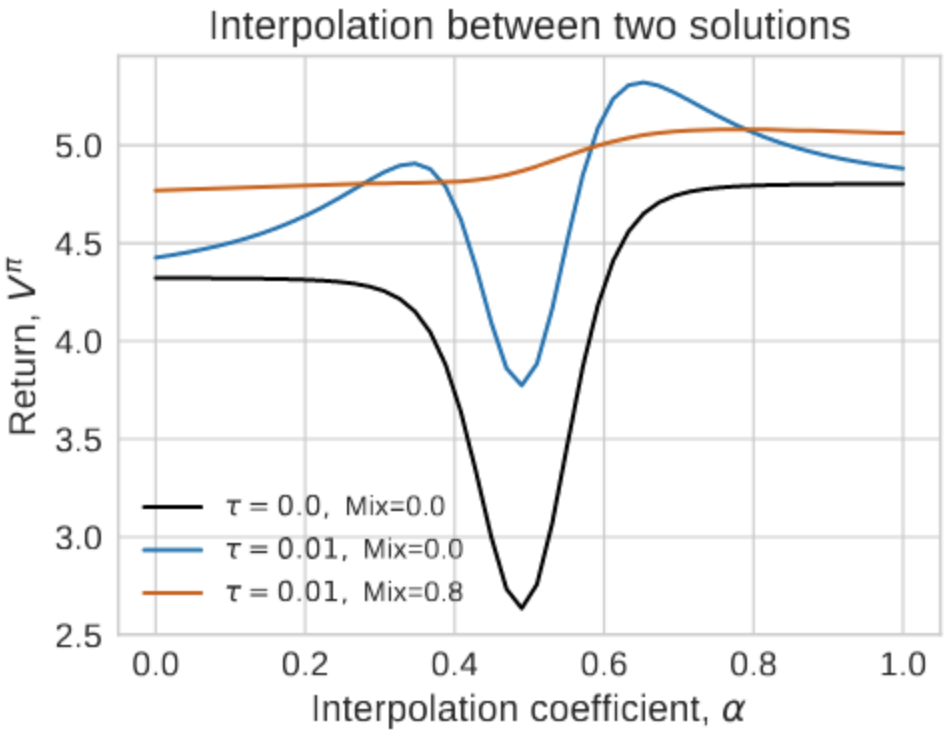
https://arxiv.org/abs/1811.11214

Machine Learning, as a discipline, continually hovers on this line between "engineering" and "science", with impulses both towards "just figure out things that work" and "actually understand the mechanics of why they work." In fairness, modern deep learning systems are complex, many-parametered beasts, that have one existence as a set of concepts, and another as a set of actual numbers, that we can only really understand from a 30,000 foot view. So, because it's hard to validate these mechanistic theories, we occasionally run into issues where, when we find something that works, we ask ourselves what would be a reasonable explanation for that working, or what would make sense, and put that forward our explanation. That, these authors argue, is what has happened in the case of using entropy augmentation in policy learning, where one commonly-understood explanation may not have as much evidence as we might light.

Basic policy optimization works by calculating expected rewards - an expectation the distribution of over all states we might arrive in, of an expectation over distributions of actions

we might take, weighted by their probability under the policy, and by their Q value - how much reward we think we'll get from taking that action, and following our current policy thereafter. We're calculating an expectation of our reward, rather than the reward or loss itself because, in reinforcement learning, we don't (generally) have visibility onto the exact reward of each action. The technique in question, of augmenting our reward with entropy, works by adding an entropy term to each action's expected reward: it calculates the entropy of our current policy's action distribution at that state, and adds a (weighted multiple of) that entropy in. Higher entropy means a policy that is closer to uniform, or less "peaky" around specific actions at a given state. By essentially adding value to our reward when our policy is high-entropy, we're incentivizing policies that are somewhat closer to uniform, at the same time that we're incentivizing policies that perform well. The canonical wisdom has always been that this is valuable because it induces more exploration, which makes some sense: a high entropy policy means that we incentivize keeping relatively more probability weight on our lowest probability actions, which would stop us from cutting those actions off as possibilities too soon. Build on top of this, there's a belief that this impacted policies because it leads to more consistent, lower-variance gradients, since you have more consistent visibility of more of the action space.

However, the authors here argue that entropy must add some value outside of this, because they construct a scenario where you're not stochastically estimating the gradient, but instead, because it's a small problem, have perfect access to it. By definition, there's no variance in the gradient estimation here, so noise reduction wouldn't help, if that were the mechanism by which entropy augmentation were useful. But they did still find it useful, a fact which they explained through investigation of the loss functions in the reasons around minimums. In their explorations, it turns out that adding entropy regularization makes objectives functions closer to being flat (this makes some sense, since if your objective was just random noise, it would be flat in every direction).



However, in the case of policy optimization, this is actually quite useful, because these areas of flatness serve to "connect" what would otherwise be valleys across which gradients would have difficulty crossing, because it would require traversing an intermediate area of low reward. (See above, where orange is the highest level of entropy weighting). Because we're adding back in reward for this high-entropy region, it gets smoothed out, and allow for gradients to escape out of local minima, into more optimal resting places.

Multi-Agent RL

Emergence of Grounded Compositional Language in Multi-Agent RL

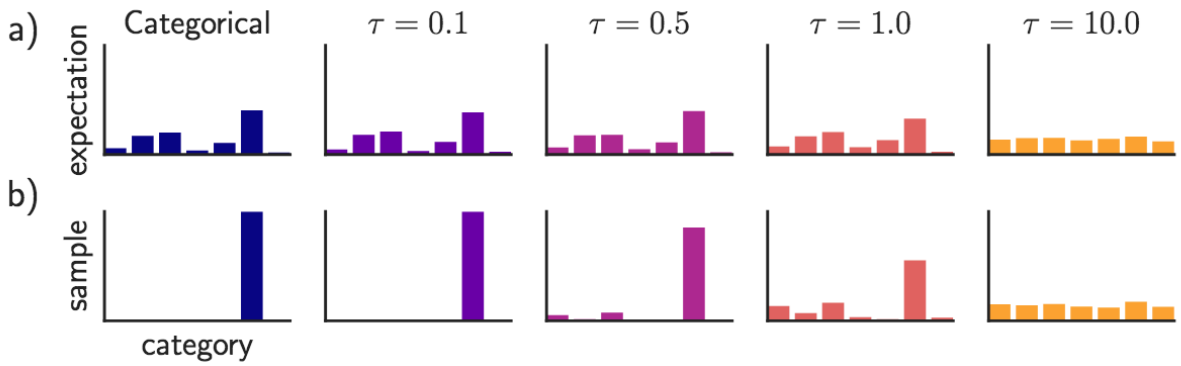
https://arxiv.org/abs/1703.04908

This paper performs a fascinating toy experiment, to try to see if something language-like in structure can be effectively induced in a population of agents, if they are given incentives that promote it. In some sense, a lot of what they find "just makes sense," but it's still a useful proof of concept to show that it can be done.

The experiment they run takes place in a simple, two-dimensional world, with a fixed number of landmarks (representing locations goals need to take place), and agents, and actions. In this construction, each agent has a set of internal goals, which can either be actions (like "go to green landmark") they themselves need to perform, or actions that they want another agent to perform. Agents' goals are not visible to other agents, but all agents' reward is defined to be the aggregated reward of all agents together, so if agent A has a goal involving an action of agent B's, it's in B's "interest" to do that action, if it can be communicated to them. In order to facilitate other agents performing goals, at each step, each agent both takes an action, and also emits an "utterance", which is just a discrete symbolic "word" out of some some fixed vocabulary of words (Note that applying "word" here is a but fuzzy; the agents do not pronounce or spell a character-based word, they just pick a discrete symbol that is playing the role of a word". Even though other agents cannot see a given agent's goals, they can see its public utterances, and so agents learn that communication is a way to induce other agents to perform desired actions.

As a mathematically interesting aside: this setup, of allowing each agent to sample a single discrete word out of a small vocabulary at each setting, takes the deployment of some interesting computational tricks to accomplish. First off, in general, sampling a discrete single symbol out of a set of possible symbols is not differentiable, since it's a discrete rather than continuous action, and derivatives require continuous functions. However, a paper from 2016 proposed a (heuristic) solution to this problem by means of the Gumbel Softmax Trick. This derives from the older "Gumbel Max Trick", which is the mathematical fact that if you want to sample from a categorical distribution, a computationally easy way to do so is to add a variable sampled from a (0,1) Gumbel distribution to the log probability of each category, and then take the argmax of this as the index of the sample category (I'm not going to go another level down into why this is true, since I think it's too far afield of the scope of this summary). Generally, argmax functions are also not differentiable. However, they can be approximated with softmaxes, which interpolate between a totally uniform and very nearly discrete-sample distribution based on a

temperature parameter. In practice, or, at least, if this paper does what the original Gumbel Softmax paper did, during training, a discrete sample is taken, but a low-temperature continuous approximation is used for actual gradient calculation (i.e. for gradients, the model pretends that it used the continuous approximation rather than the discrete sample).



Coming back to the actual communication problem, the authors do find that under these (admittedly fairly sanitized and contrived) circumstances, agents use series of discrete symbols to communicate goals to other agents, which ends up looking a lot like a very simple language. As one might expect, in environments where there were only two agents, there was no symbol that ended up corresponding to "red agent" or "blue agent", since each could realize that the other was speaking to it. However, in three-agent environments, the agents did develop symbols that clearly mapped to these categories, to specify who directions were being given to. The authors also tried cutting off verbal communication; in these situations, the agents used gaze and movement to try to signal what they wanted other agents to do. Probably most entertainingly, when neither verbal nor visual communication was allowed, agents would move to and "physically" push other agents to the location where their action needed to be performed.

Intrinsic Social Motivation via Causal Influence

https://arxiv.org/abs/1810.08647

This paper builds very directly on the idea of "empowerment" as an intrinsic reward for RL agents. Where empowerment incentivizes agents to increase the amount of influence they're able to have over the environment, "social influence," this paper's metric, is based on the degree which the actions of one agent influence the actions of other agents, within a multi-agent setting. The goals between the two frameworks are a little different. The notion of "empowerment" is built around a singular agent trying to figure out a short-term proxy for likelihood of long-term survival (which is a feedback point no individual wants to hit). By contrast, the problems that the authors of this paper seek to solve are more explicitly multi-agent coordination problems: prisoner's dilemma-style situations where collective reward requires cooperation. However, they share a mathematical basis: the idea that an agent's influence on some other element of its

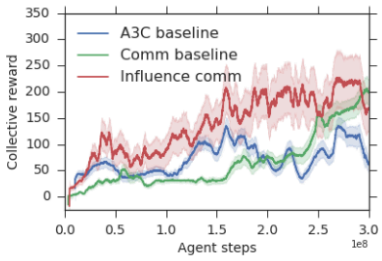
environment (be it the external state, or another agent's actions) is well modeled by calculating the mutual information between its agents and that element.

While this is initially a bit of an odd conceptual jump, it does make sense: if an action can give statistical information to help you predict an outcome, it's likely (obviously not certain, but likely) that that action influenced that outcome. In a multi-agent problem, where cooperation and potentially even communication can help solve the task, being able to influence other agents amounts to "finding ways to make oneself useful to other agents", because other agents aren't going to change behavior based on your actions, or "listen" to your "messages" (in the experiment where a communication channel was available between agents) if these signals don't help them achieve *their* goals. So, this incentive, to influence the behavior of other (self-interested) agents, amounts to a good proxy for incentivizing useful cooperation.

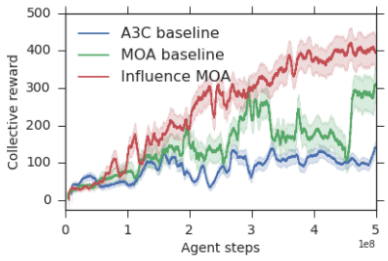
Zooming in on the exact mathematical formulations (which differ slightly from, though they're in a shared spirit with, the empowerment math): the agent's (A's) Causal Influence reward is calculated by taking a KL divergence between the action distribution of the other agent (B) conditional on the action A took, compared to other actions A might have taken. (see below. Connecting back to empowerment: Mutual Information is just the expected value of this quantity, taken over A's action distribution).

$$I_t^A = D_{KL}\left[p(a_t^B \mid a_t^A, z_t) \middle\| \sum_{\tilde{a}_t^A} p(a_t^B \mid z_t, \tilde{a}_t^A) p(\tilde{a}_t^A \mid z_t)\right] = D_{KL}\left[p(a_t^B \mid a_t^A, z_t) \middle\| p(a_t^B \mid z_t)\right].$$

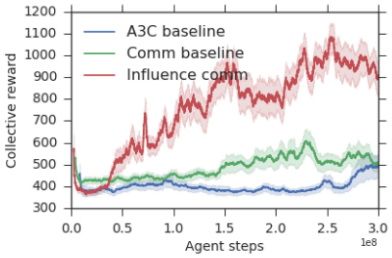
One thing you may notice from the above equation is that, because we're working in KL divergences, we expect agent A to have access to the full distribution of agent B's policy conditional on A's action, not just the action B actually took. We also require the ability to sample "counterfactuals," i.e. what agent B would have done if agent A had done something differently. Between these two requirements. If we take a realistic model of two agents interacting with each other, in only one timeline, only having access to the external and not internal parameters of the other, it makes it clear that these quantities can't be pulled from direct experience. Instead, they are calculated by using an internal model: each agent builds its own MOA (Model of Other Agents), where they build a predictive model of what an agent will do at a given time, conditional on the environment and the actions of all other agents. It's this model that is used to sample the aforementioned counterfactuals, since that just involves passing in a different input. I'm not entirely sure, in each experiment, whether the MOAs are trained concurrent with agent policies, or in a separate prior step.



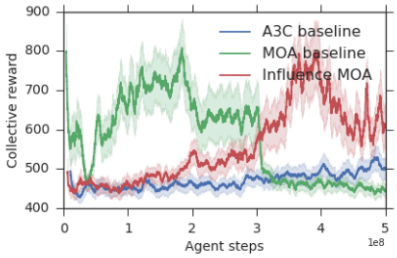
(b) *Cleanup* - Communication



(c) *Cleanup* - model of other agents



(e) *Harvest* - Communication



(f) *Harvest* - Model of other agents

Testing on, again, Prisoner's Dilemma style problems, the authors did find higher performance using their method, compared to approaches where each agent just maximizes its own external reward (which, it should be said, does depend on other agents' actions), with no explicit incentive towards collaboration. Interestingly, when they specifically tested giving agents access to a "communication channel" (the ability to output discrete signals or "words" visible to other agents), they found that it was able to train just as effectively with only an influence reward, as it was with both an influence and external reward.

Relational Forward Models for Multi Agent Learning

https://arxiv.org/abs/1809.11044

One of the dominant narratives of the deep learning renaissance has been the value of well-designed inductive bias - structural choices that shape what a model learns. The biggest example of this can be found in convolutional networks, where models achieve a dramatic parameter reduction by having features maps learn local patterns, which can then be re-used across the whole image. This is based on the prior belief that patterns in local images are generally locally contiguous, and so having feature maps that focus only on small (and gradually larger) local areas is a good fit for that prior. This paper operates in a similar spirit, except its input data isn't in the form of an image, but a graph: the social graph of multiple agents operating within a Multi Agent RL Setting. In some sense, a graph is just a more general form of a pixel image: where a pixel within an image has a fixed number of neighbors, which have fixed discrete relationships to it (up, down, left, right), nodes within graphs have an arbitrary number of nodes, which can have arbitrary numbers and types of attributes attached to that relationship.

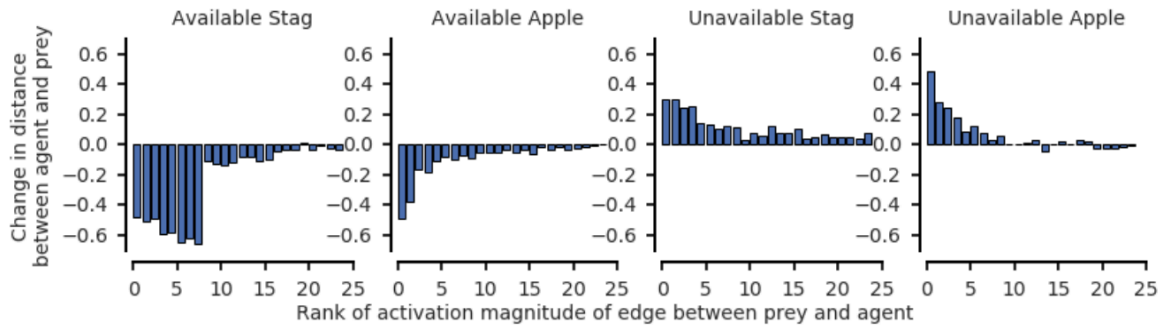
The authors of this paper use graph networks as a sort of auxiliary information processing system alongside a more typical policy learning framework, on tasks that require group coordination and knowledge sharing to complete successfully. For example, each agent might be rewarded based on the aggregate reward of all agents together, and, in the stag hunt, it might require collaborative effort by multiple agents to successfully "capture" a stag. Because of this, you might imagine that it would be valuable to be able to predict what other agents within the game are going to do under certain circumstances, so that you can shape your strategy accordingly.

The graph network used in this model represents both agents and objects in the environment as nodes, which have attributes including their position, whether they're available or not (for capture-able objects), and what their last action was. As best I can tell, all agents start out with directed connections going both ways to all other agents, and to all objects in the environment, with the only edge attribute being whether the players are on the same team, for competitive environments. Given this setup, the graph network works through a sort of "diffusion" of information, analogous to a message passing algorithm. At each iteration (analogous to a layer), the edge features pull in information from their past value and sender and receiver nodes, as well as from a "global feature". Then, all of the nodes pull in information from their edges, and their own past value. Finally, this "global attribute" gets updated based on summations over the newly-updated node and edge information. (If you were predicting attributes that were graph-level attributes, this global attribute might be where you'd do that prediction. However, in this case, we're just interested in predicting agent-level actions).

$$
\begin{aligned}
e'_k &= \phi^e\left(e_k, v_{r_k}, v_{s_k}, u\right), & \bar{e}'_i &= \rho^{e\to v}\left(E'_i\right), \\
v'_i &= \phi^v\left(\bar{e}'_i, v_i, u\right), & \bar{v}' &= \rho^{v\to u}\left(V'\right), \\
u' &= \phi^u\left(\bar{e}', \bar{v}', u\right), & \bar{e}' &= \rho^{e\to u}\left(E'\right)
\end{aligned}
$$

All of this has the effect of explicitly modeling agents as entities that both have information, and have connections to other entities. One benefit the authors claim of this structure is that it allows them more interpretability: when they "play out" the values of their graph network, which they call a Relational Forward Model or RFM, they observe edge values for two agents go up if those agents are about to collaborate on an action, and observe edge values for an agent and an object go up before that object is captured. Because this information is carefully shaped and structured, it makes it easier for humans to understand, and, in the tests the authors ran, appears to also help agents do better in collaborative games.

(a) Edge activation magnitude is predictive of future behavior.



While I find graph networks quite interesting, and multi-agent learning quite interesting, I'm a little more uncertain about the inherent "graphiness" of this problem, since there aren't really meaningful inherent edges between agents. One thing I am curious about here is how methods like these would work in situations of sparser graphs, or, places where the connectivity level between a node's neighbors, and the average other node in the graph is more distinct. Here, every node is connected to every other node, so the explicit information localization function of graph networks is less pronounced. I might naively think that - to whatever extent the graph is designed in a way that captures information meaningful to the task - explicit graph methods would have an even greater comparative advantage in this setting.

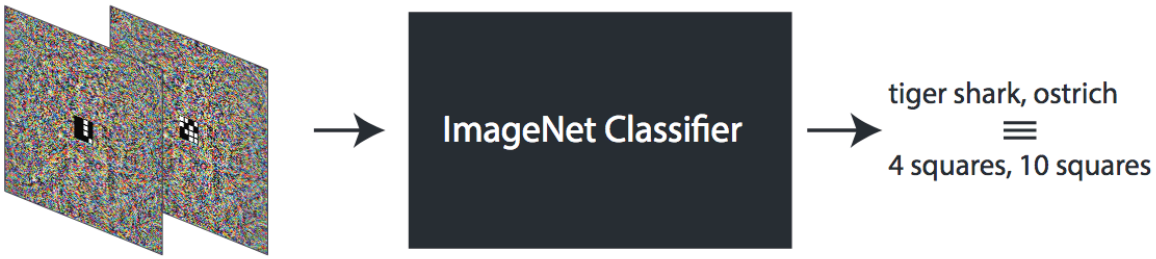
Adversarial Examples

Adversarial Reprogramming of Neural Networks

https://arxiv.org/abs/1806.11146

In the literature of adversarial examples, there's this (to me) constant question: is it the case that adversarial examples are causing the model to objectively make a mistake, or just displaying behavior that is deeply weird, and unintuitive relative to our sense of what these models "should" be doing. A lot of the former question seems to come down to arguing over about what's technically "out of distribution", which has an occasional angels-dancing-on-a-pin quality, but it's pretty unambiguously clear that the behavior displayed in this paper is weird, and beyond what I naively expected a network to be able to be manipulated to do. That said, once you're looking at it in the right frame, it seems reasonable and straightforward.

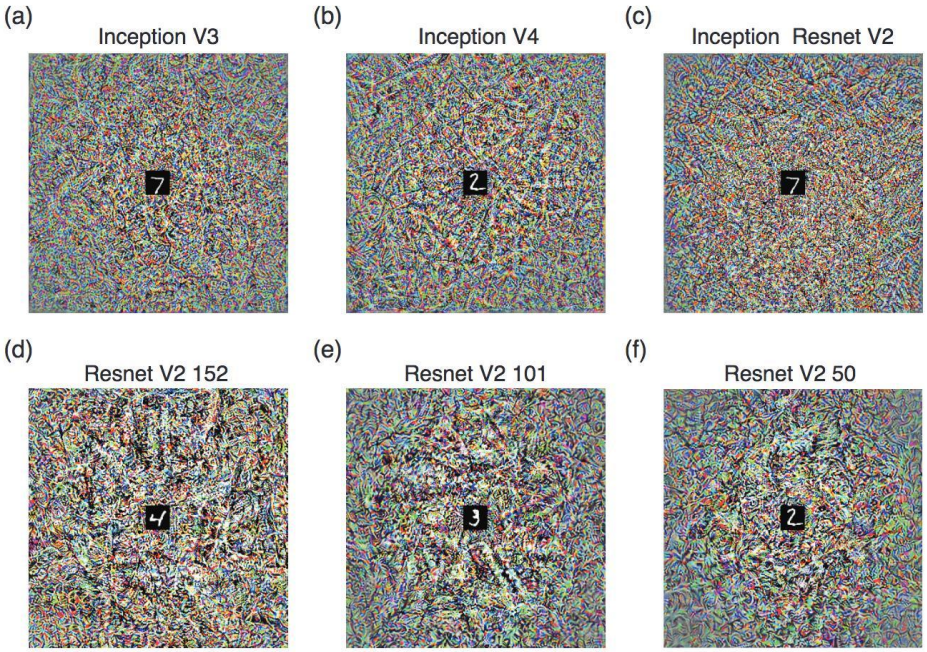
The goal these authors set for themselves is what they call "reprogramming" of a network; they want the ability to essentially hijack the network's computational engine to perform a different task, predicting a different set of labels, on a different set of inputs than the ones the model was trained on. For example, one task they perform is feeding in MNIST images at the center of a bunch of (what appear to be random, but are actually carefully optimized) pixels, and getting a network that can predict MNIST labels out the other end. Obviously, it's not literally possible to change the number of outputs that a network produces once it's trained, so the authors would arbitrarily map ImageNet outputs to MNIST categories (like, "when this model predicts Husky, that actually means the digit 7") and then judge how well this mapped output performs as a MNIST classifier. I enjoyed the authors' wry commentary here about the arbitrariness of the mapping, remarking that "a 'White Shark' has nothing to do with counting 3 squares in an image, and an 'Ostrich' does not at all resemble 10 squares".



This paper assumes a white box attack model, which implies visibility of all of the parameters, and ability to directly calculate gradients through the model. So, given this setup of a input

surrounded by modifiable pixel weights, and a desire to assign your "MNIST Labels" correctly, this becomes a straightforward optimization problem: modify the values of your input weights so as to maximize your MNIST accuracy. An important point to note here is that the same input mask of pixel values is applied for every new-task image, and so these values are optimized over a full training set of inserted images, the way that normal weights would be. One interesting observation the authors make is that, counter to the typical setup of adversarial examples, this attack would not work with a fully linear model, since you actually need your "weights" to interact with your "input", which is different each time, but these are both just different areas of your true input. This need to have different regions of input determine how other areas of input are processed isn't possible in a linear model where each input has a distinct impact on the output, regardless of other input values. By contrast, when you just need to optimize a single perturbation to get the network to jack up the prediction for one class, that can be accomplished by just applying a strong enough bias everywhere in the input, all pointing in the same direction, which can be added together linearly and still get the job done.

The authors are able to perform MNIST and the task of "count the squares in this small input" to relatively high levels of accuracy. They perform reasonably on CIFAR (as well as a fully connected network, but not as well as a convnet). They found that performance was higher when using a pre-trained ImageNet, relative to just random weights. There's some suggestion made that this implies there's a kind of transfer learning going on, but honestly, this is weird enough that it's hard to say.



One minor quibble I have with the framing of this paper is that I feel like it's stretching the original frame of "adversarial example" a bit too far, to the point of possible provoking confusion. It's not obvious that the network is making a mistake, per se, when it classifies this very out-of-

distribution input as something silly. I suppose, in an ideal world, we may want our models to return to something like a uniform-over-outputs state of low confidence when predicting out of distribution, but that's a bit different than seeing a gibbon in a picture of a panda. I don't dispute the authors claim that the behavior they're demonstrating is a vulnerability in terms of its ability to let outside actors "hijack" networks compute, but I worry we might be overloading the "adversarial example" to cover too many types of network failure modes.

On the Intriguing Connections of Regularization, Input Gradients and Transferability of Evasion and Poisoning Attacks

https://arxiv.org/abs/1809.02861

This paper focuses on the well-known fact that adversarial examples are often transferable: that is, that an adversarial example created by optimizing loss on a surrogate model trained on similar data can often still induce increased loss on the true target model, though typically not to the same magnitude as an example optimized against the target itself. Its goal is to come up with clearer theoretical formulation for transferred examples, and more clearly understand what kinds of models transfer better than others.

The authors define their two scenarios of interest as white box (where the parameters of the target model are known), and limited knowledge, or black box, where only the data type and feature representation is known, but the exact training dataset is unknown, as well as the parameters of the target model. Most of the mathematics of this paper revolve around this equation, which characterizes how to find a delta to maximize loss on the surrogate model:

$$\max_{\|\boldsymbol{\delta}\|_p \leq \varepsilon} \boldsymbol{\delta}^\top \nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \hat{\boldsymbol{w}}) = \varepsilon \|\nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \hat{\boldsymbol{w}})\|_q \,,$$

In words: you're finding a delta (perturbations of each input value) such that the p-norm of delta is less than some radius epsilon, and such that delta maximizes the dot product between delta and the model gradient with respect to the inputs. The closer two vectors are to one another, the higher their dot product. So, having your delta just *be* the model gradient w.r.t inputs maximizes that quantity. However, we also need to meet the requirement of having our perturbation's norm be less than epsilon, so we in order to find the actual optimal value, we divide by the norm of the gradient (to get ourselves a norm of 1), and multiply by epsilon (to get ourselves a norm of epsilon). This leads to the optimal value of delta being, for a norm of 2:

$$\hat{\boldsymbol{\delta}} = \varepsilon \frac{\nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \hat{\boldsymbol{w}})}{\|\nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \hat{\boldsymbol{w}})\|_2}$$

An important thing to remember is that all of the above has been using w-hat, meaning it's been an examination of what the optimal delta is when we're calculating against the surrogate model. But, if we plug in the optimal transfer value of delta we found above, how does this compare to the increase in loss if we were able to optimize against the true model?

$$\Delta \ell = \varepsilon \frac{\nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \hat{\boldsymbol{w}})}{\|\nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \hat{\boldsymbol{w}})\|_2}^{\top} \nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \boldsymbol{w}) \leq \varepsilon \|\nabla_{\boldsymbol{x}} \ell(y, \boldsymbol{x}, \boldsymbol{w})\|_2$$

Loss on the true model is, as above, calculated as the dot product of the delta perturbation with the gradient w.r.t inputs of the true model. Using the same logic as above, this quantity is maximized when our perturbation is as close as possible to the target model's gradient vector. So, the authors show, the degree to which adversarial examples calculated on one model transfer to another is mediated by the cosine distance between surrogate model's gradient vector and the target model's one. The more similar these gradients w.r.t the input are to one another, the closer surrogate-model loss increase will be to target-model loss increase. This is one of those things that makes sense once it's laid out, but it's still useful to have a specific conceptual quality to point to when predicting whether adversarial examples will transfer, rather than just knowing that they do, at least some of the time, to at least some extent.

Another interesting thing to notice from the above equation, though not directly related to transfer examples, is the right hand of the equation, the upper bound on loss increase, which is the p-norm of the gradient vector of the target model. In clearer words, this means that the amount of loss that it's possible to induce on a model using a given epsilon of perturbation is directly dependent on the norm of that model's gradient w.r.t inputs. This suggests that more highly regularized models, which are by definition smoother and have smaller gradients with respect to inputs, will be harder to attack. This hypothesis is borne out by the authors' experiments. However, they also find, consistent with my understanding of prior work, that linear models are harder to attack than non-linear ones. This draws a line between two ways we're used to thinking about model complexity/simplicity: having a less-smooth function with bigger gradients increases your vulnerability, but having nonlinear model structure seems to decrease it.

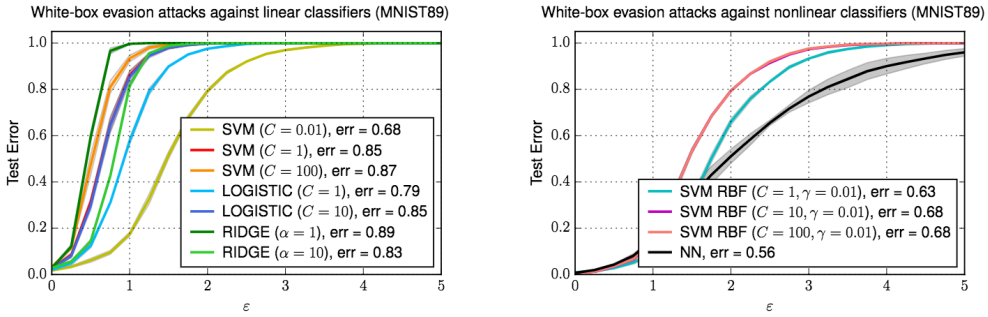


Figure 2: White-box evasion attacks on MNIST89. The reported security evaluation curves show how the test error varies against an increasing maximum admissible perturbation $\varepsilon \in [0, 5]$. The mean test error computed over the whole security evaluation curve is reported as *err* in the legend for linear (left) and nonlinear (right) classifiers.

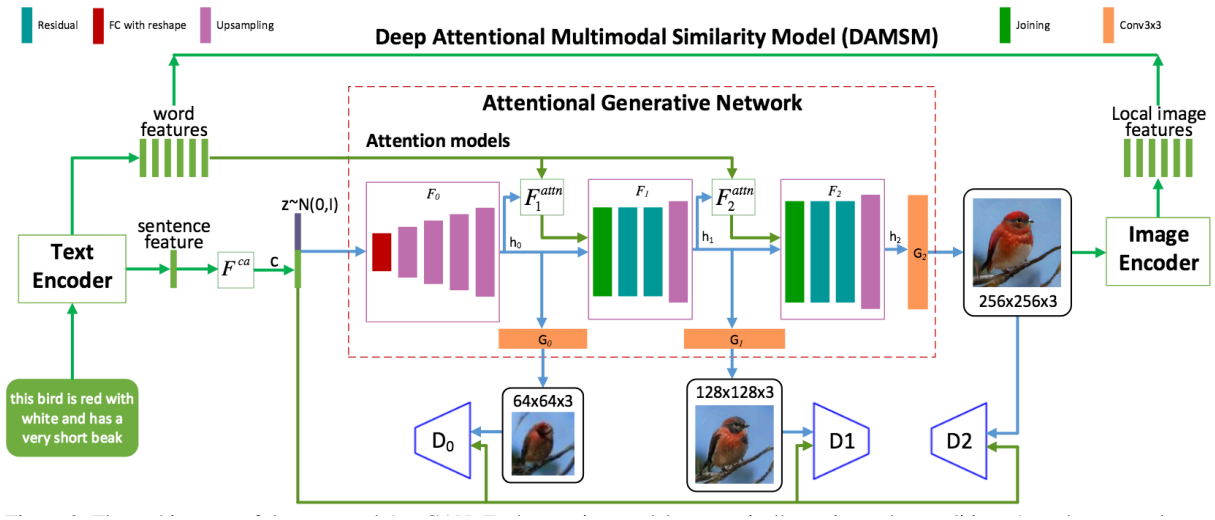
One final intriguing empirical finding of this paper is that, in addition to being the hardest models to attack when they are the target, highly regularized models work the best as surrogate models. There's a simplistic way in which this makes sense, in that if you create your examples against a "harder" adversary to begin with, they'll be in some sense stronger, and transfer better. However, I'm not sure that intuition is a correct one here.

Miscellaneous

Test to Image Generation With AttnGAN

https://arxiv.org/abs/1711.10485

This paper feels a bit like watching a 90's show, and everyone's in denim and miniskirts, except it's a 2017 ML paper, and everything uses attention. (I'll say it again, ML years are like dog years, but more so). That said, that's not a critique of the paper: finding clever ways to cobble together techniques for your application can be an important and valuable contribution. This paper addresses the problem of text to image generation: how to take a description of an image and generate an image that matches it, and it makes two main contributions: 1) a GAN structure that seems to merge insights from Attention and Progressive GANs in order to select areas of the sentence to inform details in specific image regions, and 2) a novel discriminator structure to evaluate whether a sentence matches an image.



Focusing on the first of these first: their generation system works by an iterative process, that gradually builds up image resolution, and also pulls specific information from the sentence to inform details in each region. The first layer of the network generates a first "hidden state" based on a compressed representation of the sentence as a whole (the final hidden state of a LSTM text encoder, I believe), as well as random noise (typical input to a GAN). Subsequent "hidden states" are calculated by calculating attention weightings between each region of the image, and each word in the sentence, and pulling together a per-region context vector based on that attention map. (As far as I understand it, "region" here refers to the fact that when you're at lower spatial

scales of what is essentially a progressive generation process, 64x64 rather than 256x256, for example, each "pixel" actually represents a larger region of the image). I'm using quotes around "hidden state" in the above paragraph because I think it's actually pretty confusing terminology, since it suggests a recurrent structure, but this model isn't actually recurrent: there's a specific set of weights for resolution block 0, and 1, and 2. This whole approach, of calculating a specific attention-weighted context vector over input words based on where you are in the generation process is very conceptually similar to the original domain of attention, where the attention query would be driven by the hidden state of the LSTM generating the translated version of some input sentence, except, here, instead of translating between languages, you're translating across mediums.

The loss for this model is a combination of per-layer loss, and a final, special, full-resolution loss. At each level of resolution, there exists a separate discriminator, which seems to be able to take in both 1) only an image, and judge whether it thinks that image looks realistic on it's own, and 2) an image and a global sentence vector, and judge whether the image matches the sentence. It's not fully clear from the paper, but it seems like this is based on just feeding in the sentence vector as additional input?

$$\mathcal{L}_{G_i} = \underbrace{-\frac{1}{2}\mathbb{E}_{\hat{x}_i \sim p_{G_i}}[\log(D_i(\hat{x}_i)]}_{\text{unconditional loss}} \underbrace{-\frac{1}{2}\mathbb{E}_{\hat{x}_i \sim p_{G_i}}[\log(D_i(\hat{x}_i, \bar{e})]}_{\text{conditional loss}},$$

For each non-final layer's discriminator, the loss is a combination of both of these unconditional and conditional losses.

The final contribution of this paper is something they call the DAMSM loss: the Deep Attention Multimodal Similarity Model. This is a fairly complex model structure, whose ultimate goal is to assess how closely a final generated image matches a sentence. The whole structure of this loss is based on projecting region-level image features (from an intermediate, 17x17 layer of a pretrained Inception Net) and word features into the same space, and then calculating dot product similarities between them, which are then used to build "visual context vectors" for each word (for each word, created a weighted sum of visual vectors, based on how similar each is to the word). Then, we take each word's context vector, and see how close it is to the original word vector. If we, again, imagine image and word vectors as being in a conceptually shared space, then this is basically saying "if I take a weighted average of all the things that are the most similar to me, how ultimately similar is that weighted average to me". This allows there to be a "concept representation" match found when, for example, a particular word's concept, like "beak", is only present in one region, but present there very strongly: the context vector will be strongly weighted towards that region, and will end up being very close, in cosine similarity terms, to the word itself. By contrast, if none of the regions are a particularly good match for the word's concept, this value will be low. DAMSM then aggregates up to an overall "relevance" score between a sentence and image, that's simply a sum over a word's "concept representation",

for each word in a sentence. It then calculates conditional probabilities in two directions: what's the probability of the sentence, given the image (relevance score of (Sent, Imag), divided by that image's summed relevance with all possible sentences in the batch), and, also, what's the probability of the image, given the sentence (relevance score of the pair, divided by the sentence's summed relevance with all possible images in the batch).

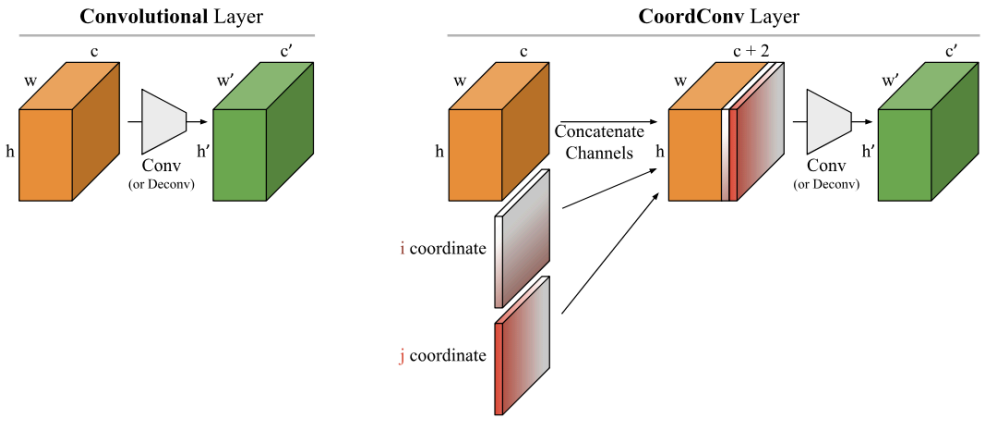
In addition to this word-level concept modeling, DAMSM also has full sentence-level versions, where it simply calculates the relevance of each (sentence, image) pair by taking the cosine similarity between the global sentence and global image features (the final hidden state of an encoder RNN, and the final aggregated InceptionNet features, respectively). All these losses are aggregated together, to get one that uses both global information, and information as to whether specific words in a sentence are represented well in an image.

An intriguing failing of convolutional neural networks and the CoordConv solution

https://arxiv.org/abs/1807.03247

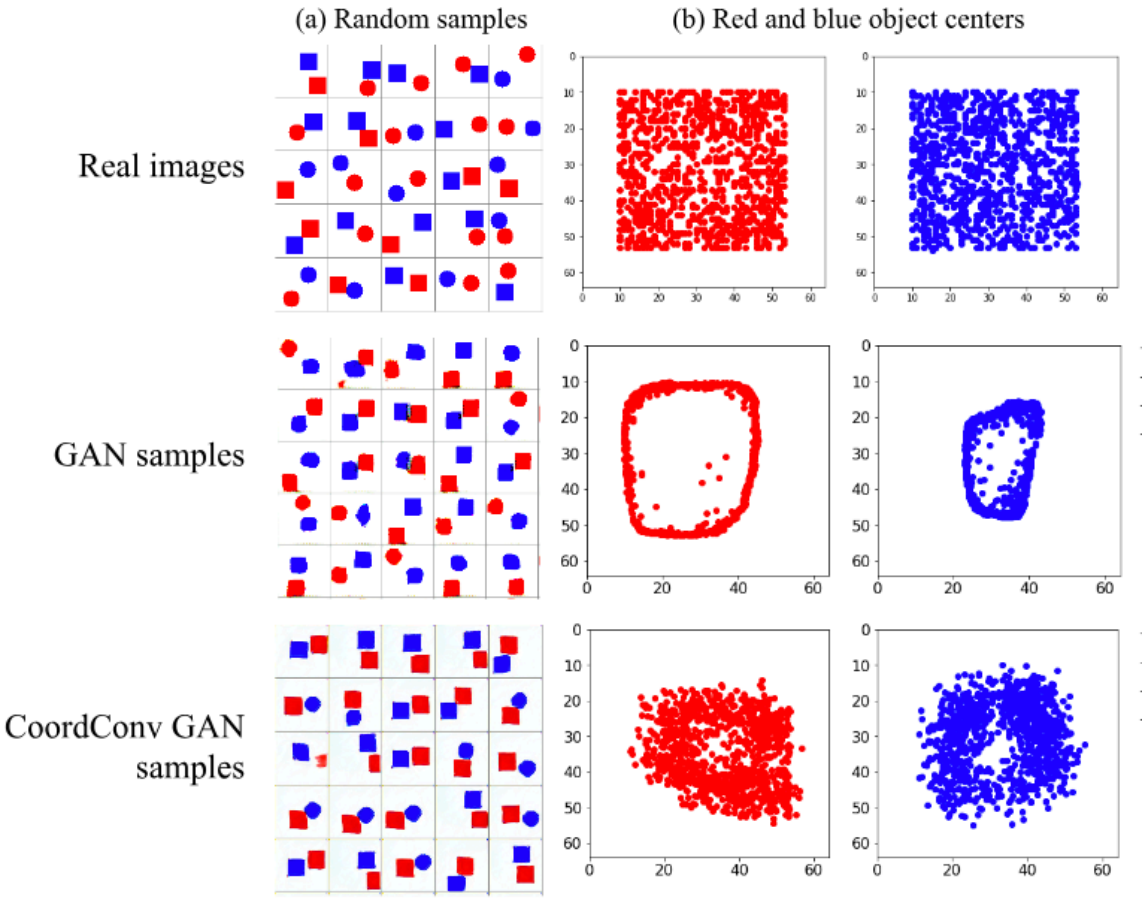
This is a paper where I keep being torn between the response of "this is so simple it's brilliant; why haven't people done it before," and "this is so simple it's almost tautological, and the results I'm seeing aren't actually that surprising". The basic observation this paper makes is one made frequently before, most recently to my memory by Geoff Hinton in his Capsule Net paper: sometimes the translation invariance of convolutional networks can be a bad thing, and lead to worse performance. In a lot of ways, translation invariance is one of the benefits of using a convolutional architecture in the first place: instead of having to learn separate feature detectors for "a frog in this corner" and "a frog in that corner," we can instead use the same feature detector, and just move it over different areas of the image. However, this paper argues, this makes convolutional networks perform worse than might naively be expected at tasks that require them to remember or act in accordance with coordinates of elements within an image.

For example, they find that normal convolutional networks take nearly an hour and 200K worth of parameters to learn to "predict" the one-hot encoding of a pixel, when given the (x,y) coordinates of that pixel as input, and only get up to about 80% accuracy. Similarly, trying to take an input image with only one pixel active, and predict the (x,y) coordinates as output, is something the network is able to do successfully, but only when the test points are sampled from the same spatial region as the training points: if the test points are from a held-out quadrant, the model can't extrapolate to the (x, y) coordinates there, and totally falls apart.



The solution proposed by the authors is a really simple one: at one or more layers within the network, in addition to the feature channels sent up from the prior layer, add two addition channels: one with a with deterministic values going from -1 (left) to 1 (right), and the other going top to bottom. This essentially adds two fixed "features" to each pixel, which jointly carry information about where it is in space. Just by adding this small change, we give the network the ability to use spatial information or not, as it sees fit. If these features don't prove useful, their weights will stay around their initialization values of expectation-zero, and the behavior should be much like a normal convolutional net. However, if it proves useful, convolution filters at the next layer can take position information into account. It's easy to see how this would be useful for this paper's toy problems: you can just create a feature detector for "if this pixel is active, pass forward information about it's spatial position," and predict the (x, y) coordinates out easily. You can also imagine this capability helping with more typical image classification problems, by having feature filters that carry with them not only content information, but information about where a pattern was found spatially.

The authors do indeed find comparable performance or small benefits to ImageNet, MNIST, and Atari RL, when applying their layers in lieu of normal convolutional layer. On GANs in particular, they find less mode collapse, though I don't yet 100% follow the intuition of why this would be the case.



Visualizing the Loss Landscape of Neural Networks

https://arxiv.org/abs/1712.09913

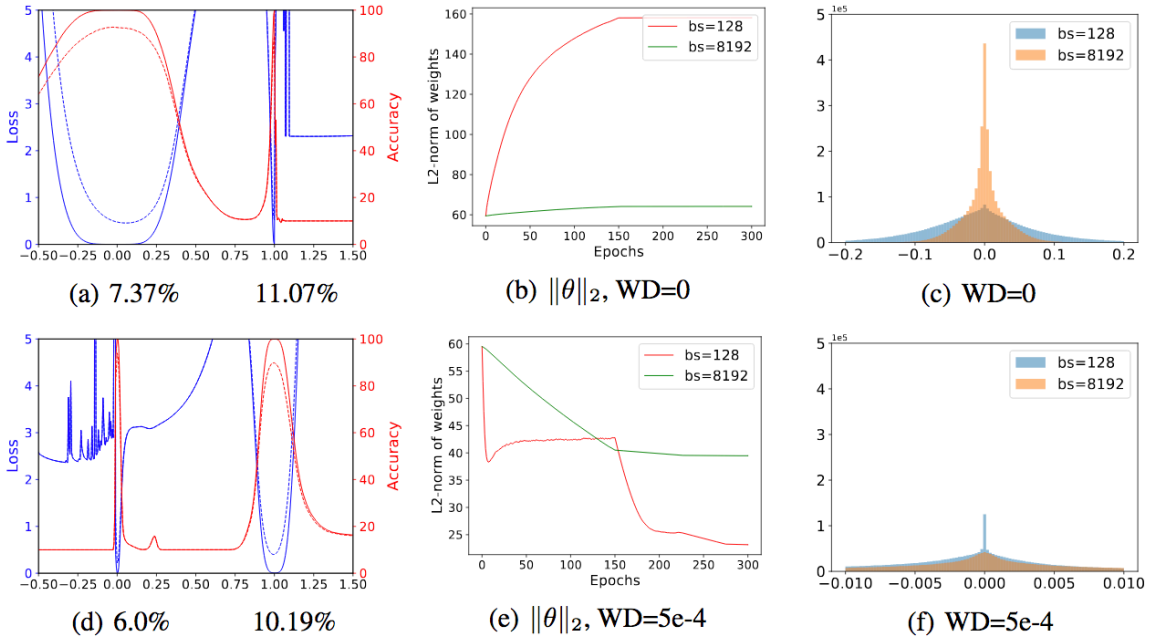
This paper was a real delight to read, and even though I'm summarizing it here, I'd really encourage you, if you're reading this, to read the paper itself, since I found it to be unusually clearly written. It tackles the problem of understanding how features of loss functions - these integral, yet arcane, objects defined in millions of parameter-dimensions - impact model performance. Loss function analysis is generally a difficult area, since the number of dimensions and number of points needed to evaluate to calculate loss are both so high. The latter presents computational challenges, the former ones of understanding: human brains and many-dimensional spaces are not a good fit. Overall, this paper contributes by 1) arguing for a new way of visualizing loss functions, 2) demonstrating how and in what cases "flatness" of loss function contributes to performance and trainability, and 3))

The authors review a few historically common ways of visualizing loss functions, before introducing their variant. The simplest, one-dimensional visualization technique, 1-D Linear

Interpolation, works by taking two parameter settings (say, a random initialization, and the final network minimum), and smoothly interpolating between the two, by taking a convex combination mediated by alpha. Then, you can plot the value of the loss at all of these parameter configurations as a function of alpha. If you want to plot in 2D, with a contour plot, you can do so in a pretty similar manner, by picking two random "direction vectors" of the same size as the parameter vector, and then adding amounts of those directions, weighted by alpha and beta, to your starting point. These random directions become your axes, and you get a snapshot of the change in your loss function as you move along them.

The authors then make the observation that these techniques can't natively be used to compare two different models, if the parameters of those models are on different scales. If you take a neural net, multiply one layer by 10, and then divide the next layer by 10, you've essentially done a no-op that won't impact the outcome. However, if you're moving by a fixed amount along your random direction in parameter space, you'll have to move much farther to go the commensurate amount of distance in the network that's been multiplied by 10. To address this problem, they suggest a simple fix: after you've selected each of your random directions, scale the value in each direction vector by the norm of the filter that corresponds to that value. This gets rid of the sensitivity of your plots to the scale of weights. (One thing I admit I'm a little confused by here is the fact that each value in the direction vector corresponds to a filter, rather than to a weight; I would have natively thought theta, and all the direction vectors, are of length number-of-model-parameters, and each value is a single weight. I think I still broadly grasp the intuition, but I'd value having a better sense of this).

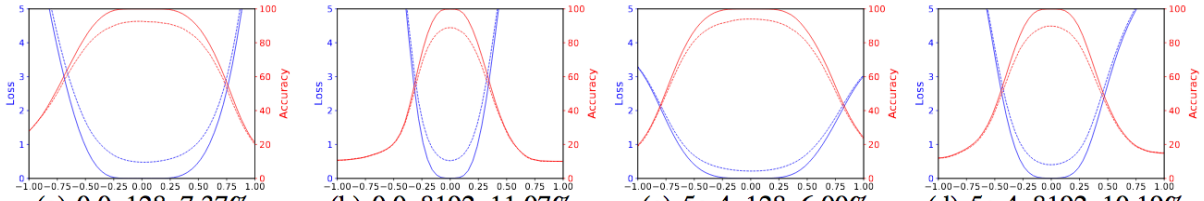
To demonstrate the value of their normalization system, they compare the interpolation plots for a model with small and large batch size, with and without weight decay. Small batches are known to increase flatness of the loss function around the eventual minimum, which seems co-occurrent with good generalization results. And, that bears out in the original model's linear interpolation, where the small model has the wider solution basin, and also better performance. However, once weight decay is applied, the small-batch basin appears to shrink to be very narrow, although small-batch still has dominant performance. At first glance, this would seem to be a contradiction of the "flatter solutions mean more generalization" rule.



(a) 7.37% 11.07% (b) $\|\theta\|_2$, WD=0 (c) WD=0

(d) 6.0% 10.19% (e) $\|\theta\|_2$, WD=5e-4 (f) WD=5e-4

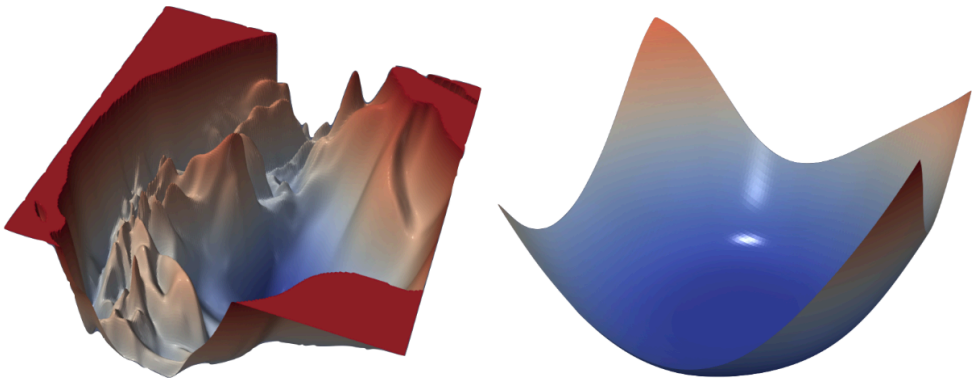
But this is just because weight decay hits smaller models more strongly, because they have more distinct updates during which they apply the weight decay penalty. This means that when weight decay is applied, the overall scale of weights in the small-batch network is lower, and so it's solution looked "sharp" when plotted on the same weight scale as the large-batch network. When normalization was used, this effect by and large went away, and you once again saw higher performance with flatter loss functions. (batch size and performance with and without weight decay, shown normalized below)



(a) 0.0, 128, 7.37% (b) 0.0, 8192, 11.07% (c) 5e-4, 128, 6.00% (d) 5e-4, 8192, 10.19%

A few other, more scattered observations from the paper:

- I've heard explanations of skip connections in terms of "giving the model shorter gradient paths between parameters and output," but haven't really seen an argument for why skip connections lead to smoother loss functions, even they empirically seem to



(a) ResNet-110, no skip connections (b) DenseNet, 121 layers

Figure 4: The loss surfaces of ResNet-110-noshort and DenseNet for CIFAR-10.

- The authors also devise a technique for visualizing the change in loss function along the trajectory taken by the optimization algorithm, so that different ones can be compared. The main problem in previous methods for this has been that optimization trajectories happen in a low-dimensional manifold within parameter space, so if you just randomly select directions, you won't see any interesting movement along the trajectory. To fix this, they choose as their axes the principal components you get from making a matrix out of the parameter values at each epoch: this prioritizes the parameters that had the most variance throughout training.

Embedding Grammars

https://arxiv.org/abs/1808.04891

This paper is, on the whole, a refreshing jaunt into the applied side of the research word. It isn't looking to solve a fundamental machine learning problem in some new way, but it does highlight and explore one potential beneficial application of a common and widely used technique: specifically, combining word embeddings with context-free grammars (such as: regular expressions), to make the latter less rigid.

Regular expressions work by specifying specific hardcoded patterns of symbols, and matching against any strings in some search set that match those patterns. They don't need to specify specific characters - they can work at higher levels of generality, like "uppercase alphabetic character" or "any character", but they're still fundamentally hardcoded, in that the designer of the expression needs to create a specification that will affirmatively catch all the desired cases. This can be a particular challenging task when you're trying to find - for example - all sentences that match the pattern of someone giving someone else a compliment. You might want to match against "I think you're smart" and also "I think you're clever". However, in the normal use of regular expressions, something like this would be nearly impossible to specify, short of writing out every synonym for "intelligent" that you can think of.

The “Embedding Grammars” paper proposes a solution to this problem: instead of enumerating a list of synonyms, simply provide one example term, or, even better, a few examples, and use those term’s word embedding representation to define a “synonym bubble” (my word, not theirs) in continuous space around those examples. This is based on the oft-remarked-upon fact that, because word embedding systems are generally trained to push together words that can be used in similar contexts, closeness in word vector space frequently corresponds to words being synonyms, or close in some other sense. So, if you “match” to any term that is sufficiently nearby to your exemplar terms, you are performing something similar to the task of enumerating all of a term’s syllables. Once this general intuition is in hand, the details of the approach are fairly straightforward: the authors try a few approaches, and find that constructing a bubble of some epsilon around each example’s word vector, and matching to anything inside that bubble, works the best as an approach.

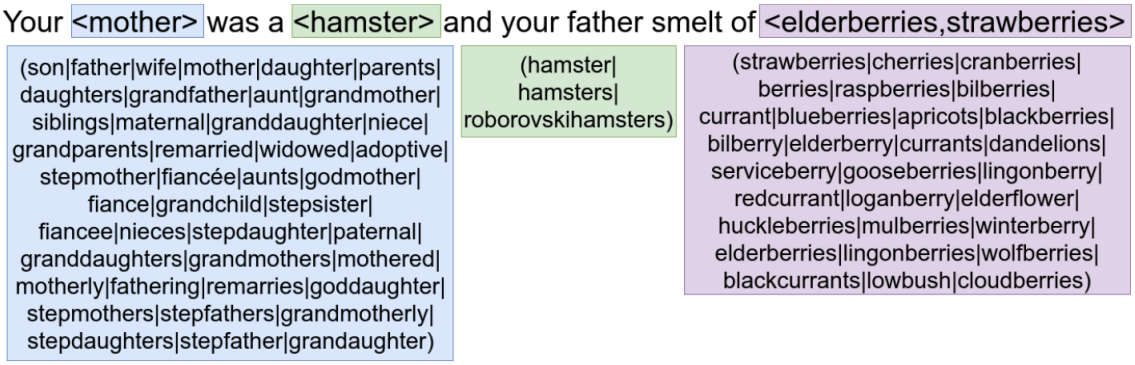


Figure 1: Sample regexv expansions obtained using the 500,000 most common tokens in the English-language fastText corpus. Matches were selected using the neighbor expansion method with ϵ=0.35.

Overall, this seems like a clever idea; one imagines that the notion of word embeddings will keep branching out into ever more far-flung application as time goes on. There are reasons to be skeptical of this paper, though. Fundamentally, word embedding space is a “here there be dragons” kind of place: we may be able to observe broad patterns, and might be able to say that “nearby words tend to be synonyms,” but we can’t give any kind of guarantee of that being the case. As an example of this problem, often the nearest thing to an example, after direct synonyms, are direct antonyms, so if you set too high a threshold, you’ll potentially match to words exactly the opposite of what you expect. We are probably still a ways away from systems like this one being broady useful, for this and other reasons, but I do think it’s valuable to try to understand what questions we’d want answered, what features of embedding space we’d want more elucidated, before applications like these would be more stably usable.

Deep Image Reconstruction from fMRI Data

https://www.biorxiv.org/content/early/2017/12/28/240317

This is a paper where I'm stuck between "this is a clever, satisfying application of a ML process I'm familiar with" and "the same size is sufficiently small, and the field one I'm sufficiently unfamiliar with, that I have a hard time judging to what extent the ML is adding value". That being given as a caveat, I still think it's valuable to understand the mechanics of what this paper is doing.

The goal of this paper is to reconstruct the visual image being seen by a subject in a fMRI scanner, based on the fMRI data they generate while looking at it. Prior work had shown an ability to predict an image out of a set of exemplar images, based on fMRI patterns, but *as far as I can tell*, this paper is the first to actually generate the full images de novo, rather than selecting them out of an existing set. It does this by plugging together multiple different "information architecture" processes that, while individually straightforward, come together into an interesting, coherent system.

- First, a predictive model is created, that learns to predict the higher-layer activations of a given neural net from fMRI activations, using a training set of image for which fMRI data was collected. (This approach was previously proposed in an earlier paper by this same group)
- Using this model, when a subject's fMRI data was read while viewing a given image, they used that model to predict (or "translate" fMRI features into predicted network activations, at one, or, in this paper's case, many, levels of the neural net hierarchy)
- Then, they use a technique you may recognize if you've read any of the style transfer literature: they perform gradient descent the pixels of a random image to try to minimize the distance between the layer activations of those pixels, and the target activations predicted from fMRI
- Additionally, they add to this loss function the loss from the discriminator of a (I think, pre-trained) GAN: this gives a "realness score," that the authors frame as a natural image prior. In general, using this prior pushes them towards more smooth, realistic images, and away from ones with odd patterns and artifacts

All in all, it's hard to say how well this approach performs, in some part because the data requires human subjects (and is thus inherently small scale), and in some part because we don't have good automated metrics for this kind of reconstruction, where we don't realistically expect to be able to get e.g. good pixel-level accuracy. That said, some interesting tidbits from the results section:

- When reconstructing natural images, it seems (very, very subjectively) like one of the highest-fidelity areas on animals is their face
- The authors tried asking subjects to imagine both natural images and letters/symbols, and while the natural images were kind of a mess (presumably because imagining, say, a leopard is a complex operation), reconstruction on the symbols was actually pretty impressively good. I'd be really curious how this behaved on people with good or bad subjective visual memory/visual reasoning abilities.

- The process of predicting layer activations from fMRI features is very much not a perfect one, and this itself could be a possible avenue for improvement (authors currently use a linear regression). Higher network layers are easier to accurately reconstruct