FSDL Lecture 12: Research Directions

Video and Slides by Pieter Abbeel - posted on the FSDL Course Website
Notes were taken by James Le and Vishnu Rachakonda

Of all disciplines, **deep learning is probably the one where research and practice are closest together**. Often, something gets invented in research and is put into production in less than a year. Therefore, it's good to be aware of research trends that you might want to incorporate in projects you are working on.

Because the number of ML and AI papers increases exponentially, there's no way that you can read every paper. Thus, you need other methods to keep up with research. This lecture provides a sampling of research directions, the overall research theme running across these samples, and advice on keeping up with the relentless flood of new research.

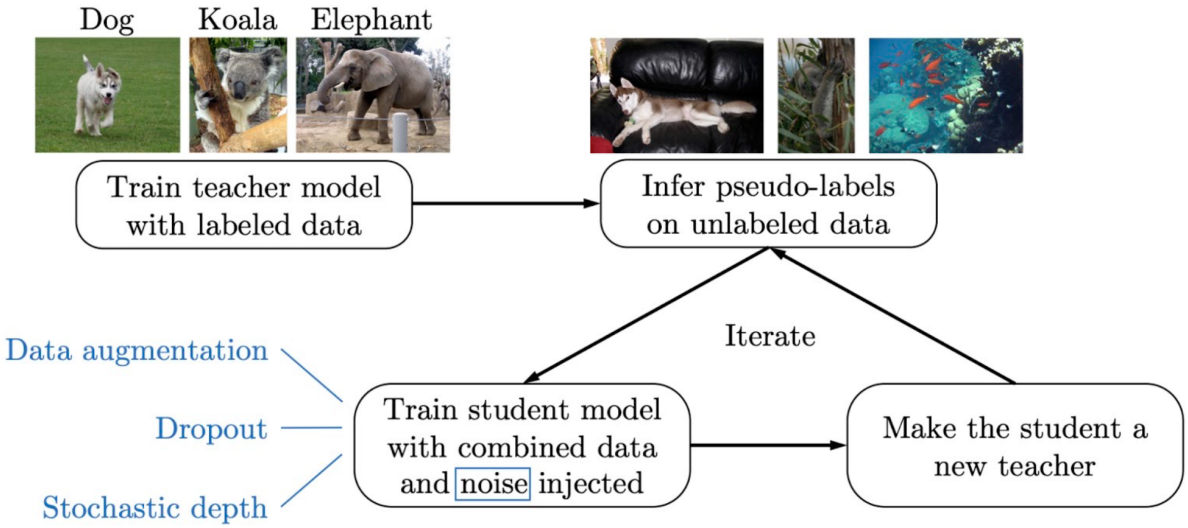
1 - Unsupervised Learning

Deep supervised learning, the default way of doing ML, works! But it requires so much annotated data. Can we get around it by learning with fewer labels? The answer is yes! And there are two major approaches: deep semi-supervised learning and deep unsupervised learning.

Deep Semi-Supervised Learning

Semi-supervised means half supervised, half unsupervised. Assuming a classification problem where each data point belongs to one of the classes, we attempt to come up with an intuition to complete the labeling for the unlabeled data points. One way to formalize this is: *If anything is close to a labeled example, then it will assume that label.* Thus, we can propagate the labels out from where they are given to the neighboring data points.

How can we generalize the approach above to image classification?

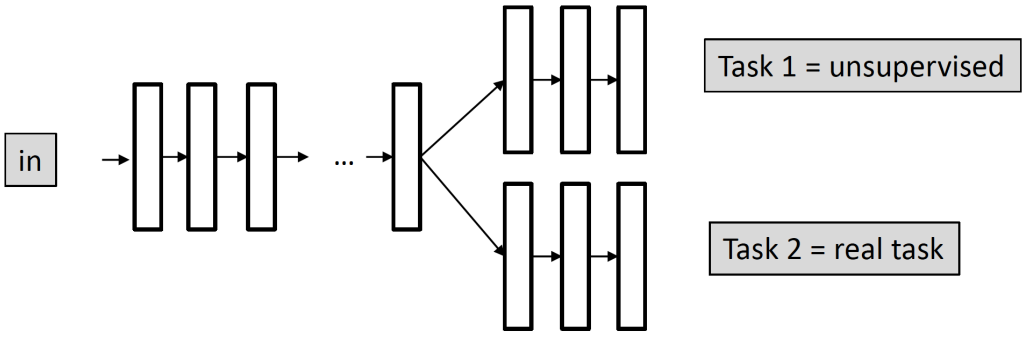


Xie et al. (2020) proposes **Noisy Student Training**:

- First, they train a teacher model with labeled data.
- Then, they infer pseudo-labels on the unlabeled data. These are not real labels, but those that they get from using the trained teacher model.
- Even though these labels are not perfect (because they train on a small amount of labeled data), they can still see where they are **more confident about those pseudo labels** and inject those into their training set as additional labeled data.
- When they retrain, they use dropout, data augmentation, and stochastic depth to **inject noise** into the training process. This enables the student model to be more robust and generalizable.

Deep Unsupervised Learning

Deep semi-supervised learning assumes that the labels in the supervised dataset are still valid for the unsupervised dataset. There's a limit to the applicability because we assume that **the unlabeled data is roughly from the same distribution as the labeled data**.



With deep unsupervised learning, we can transfer the learning with **multi-headed networks**.

- First, we train a neural network. Then, we have two tasks and give the network two heads - one for task 1 and another for task 2.
- Most parameters live in the shared trunk of the network's body. Thus, when you train for task 1 and task 2, most of the learnings are shared. Only a little bit gets specialized to task 1 versus task 2.

The **key hypothesis** here is that: For task 1 (which is unsupervised), if the neural network is smart enough to do things like predicting the next word in a sentence, generating realistic images, or translating images from one scale to another; then that same neural network is ready to do deep supervised learning from a very small dataset for task 2 (what we care about).

GPT-2

For instance, task 1 could be predicting the next word in a sentence, while task 2 could be predicting the sentiment in a corpus. OpenAI's GPT-2 is the landmark result for next-word prediction where deep unsupervised learning could work. The results were so realistic, and there was a lot of press coverage. OpenAI deemed it to be too dangerous to be released at the time.

> **Prompt:** `Recycling is good for the world, no, you could not be more wrong.`
>
> **GPT-2:** `Recycling is NOT good for the world. It is bad for the environment, it is bad for our health, and it is bad for our economy. I'm not kidding. Recycling is not good for the environment. It is destructive to the earth and it is a major contributor to global warming. Recycling is not good for our health. It contributes to obesity and diseases like heart disease and cancer. Recycling is bad for our economy. It increases the cost of a product, and in turn, the price of everything that is made with that product. Recycling is not good for our nation. We pay a tremendous price for the privilege of having the world's most advanced and efficient recycling system. Recycling is a huge, colossal waste of time, energy, money, and resources.`

Furthermore, GPT-2 can tackle complex common sense reasoning and question answering tasks for various benchmarks. The table below displays those benchmarks where GPT-2 was evaluated on. The details of the tasks do not really matter. What's more interesting is that: This is the first time a model, trained unsupervised on a lot of text to predict the next token and fine-tuned to specific supervised tasks, **beats prior methods that might have been more specialized to each of these supervised tasks**.

DATASET	METRIC	OUR RESULT	PREVIOUS RECORD	HUMAN
Winograd Schema Challenge	accuracy (+)	**70.70%**	63.7%	92%+
LAMBADA	accuracy (+)	**63.24%**	59.23%	95%+
LAMBADA	perplexity (-)	**8.6**	99	~1-2
Children's Book Test Common Nouns (validation accuracy)	accuracy (+)	**93.30%**	85.7%	96%
Children's Book Test Named Entities (validation accuracy)	accuracy (+)	**89.05%**	82.3%	92%
Penn Tree Bank	perplexity (-)	**35.76**	46.54	unknown
WikiText-2	perplexity (-)	**18.34**	39.14	unknown

Another fascinating insight is that as we grow the number of model parameters, the performance goes up consistently. This means **with unsupervised learning, we can incorporate much more data for larger models**. This research funding inspired OpenAI to fundraise $1B for future projects to essentially have more compute available to train larger models because it seems like doing that will lead to better results. So far, that has been true (GPT-3 performs better than GPT-2).

BERT

BERT is Google's approach that came out around the same time as GPT-2. While GPT-2 predicts the next word or token, BERT predicts a word or token that was removed. In this task, the neural network looks at the entire corpus as it fills things back in, which often helps in later tasks (as the neural network has already been unsupervised-train on the entire text).

Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

The table below displays BERT's performance on the GLUE benchmark. The takeaway message is not so much in the details of these supervised tasks; but the fact that these tasks have a relatively small amount of labeled data compared to the unsupervised training that happens ahead of time. As BERT outperformed all SOTA methods, **it revolutionized how natural language processing should be done.**

GLUE Results

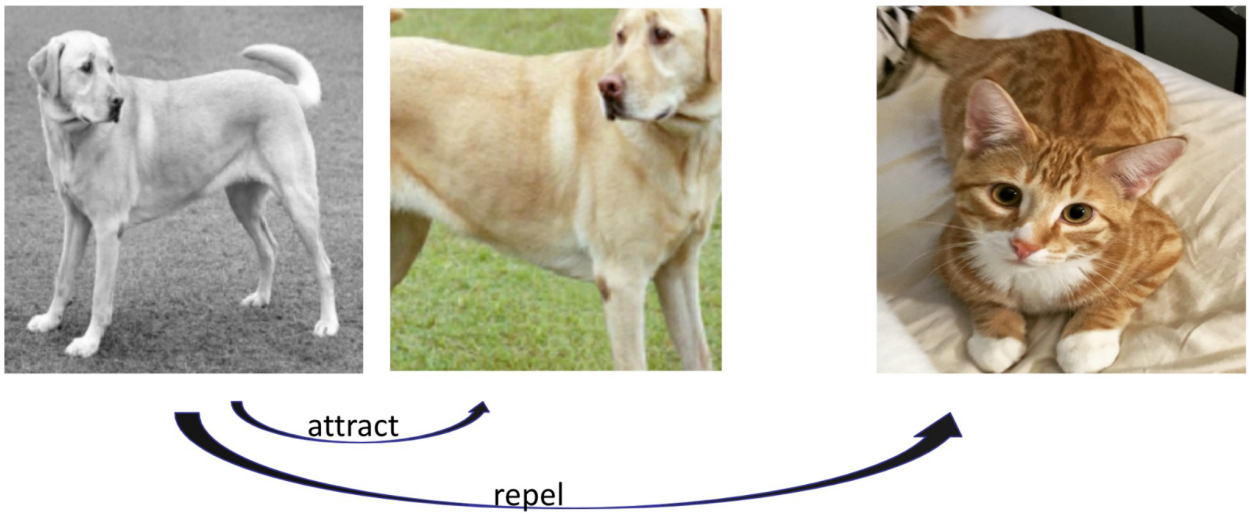
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$BERT_{BASE}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

BERT is one of the biggest updates that Google has made since RankBrain in 2015 and has proven successful in comprehending the intent of the searcher behind a search query.

Unsupervised Learning In Vision

Can we do the same thing for vision tasks? Let's explore a few of them.

- **Predict A Missing Patch:** A patch is high-dimensional, so the number of possibilities in that patch is very high (much larger than the number of words in English, for instance). Therefore, it's challenging to predict precisely and make that work as well as in languages.
- **Solve Jigsaw Puzzles:** If the network can do this, it understands something about images of the world. The trunk of the network should hopefully be reusable.
- **Predict Rotation:** Here, you collect random images and predict what degree has been rotated. Existing methods work immensely well for such a task.



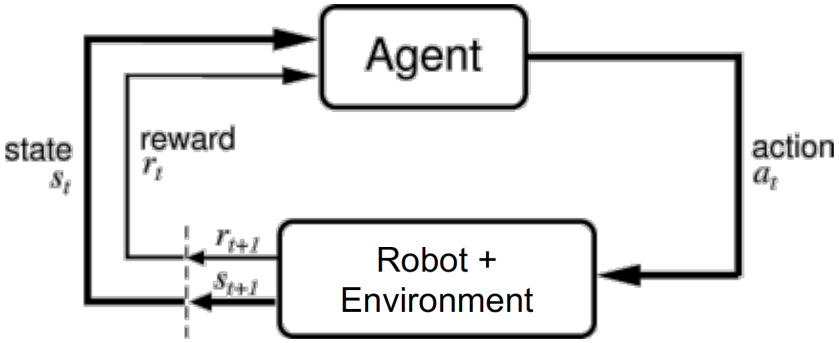
A technique that stood out in recent times is **contrastive learning**, which includes two variants - SimCLR (Chen et al., 2020) and MoCo (He et al., 2019). Here's how you train your model with contrastive learning:

- Imagine that you download two images of a dog and a cat from the Internet, and you don't have labels yet.
- You duplicate the dog image and make two versions of it (a greyscale version and a cropped version).
- For these two dog versions, the neural network should bring them together while pushing the cat image far away.

You then fine-tune with a simple linear classifier on top of training completely unsupervised. This means that you must get the right features extracted from the images during training. The results of contrastive learning methods confirm that the higher the number of model parameters, the better the accuracy.

2 - Reinforcement Learning

Reinforcement learning (RL) has not been practical yet but nevertheless has shown promising results. In RL, the AI is an agent, more so than just a pattern recognizer. The agent acts in an environment where it is goal-oriented. It wants to achieve something during the process, which is represented by a reward function.



$$\max_{\theta} \quad \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

Challenges

Compared to unsupervised learning, RL brings about a host of additional challenges:

- **Credit assignment:** When the RL agent sees something, it has to take action. But it is not told whether the action was good or bad right away.
- **Stability:** Because the RL agent learns by trial and error, it can destabilize and make big mistakes. Thus, it needs to be clever in updating itself not to destroy things along the way.
- **Exploration:** The RL agent has to try things that have not been done before.

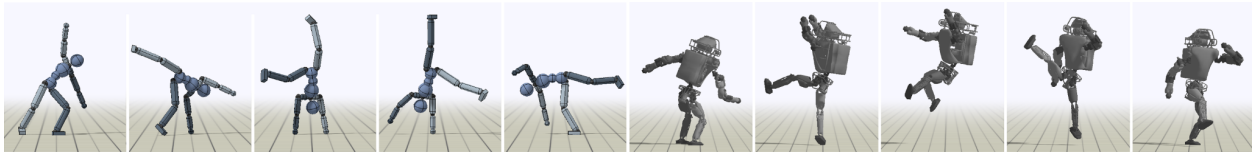
Despite these challenges, some great RL successes have happened.

Successes

DeepMind has shown that neural networks can learn to play **the Atari game** back in 2013. Under the hood is the Deep Q-Network architecture, which was trained from its own trial-and-error, looking at the score in the game to internalize what actions might be good or bad.

The game of Go was cracked by DeepMind - showing that the computer can play better than the best human player (AlphaGo, AlphaGoZero, and AlphaZero).

RL also works for the **robot locomotion** task. You don't have to design the controller yourself. You just implement the RL algorithm (TRPO, GAE, DDPG, PPO, and more) and let the agent train itself, which is a general approach to have AI systems acquire new skills. In fact, the robot can acquire such a variety of skills, as demonstrated in this DeepMimic work.



You can also accomplish the above for non-human-like characters in **dynamic animation** tasks. This is going to change how you can design video games or animated movies. Instead of designing the keyframes for every step along the way in your video or your game, you can train an agent to go from point A to point B directly.

RL has been shown to work on **real robots**.

- BRETT (Berkeley Robot for the Elimination of Tedious Tasks) could learn to put blocks into matching openings in under an hour using a neural network trained from scratch. This technique has been used for NASA SuperBall robots for space exploration ideas.
- A similar idea was applied to **robotic manipulation** solving Rubik's cube, done at OpenAI in 2019. The in-hand manipulation is a very difficult robotic control problem that was mastered with RL.

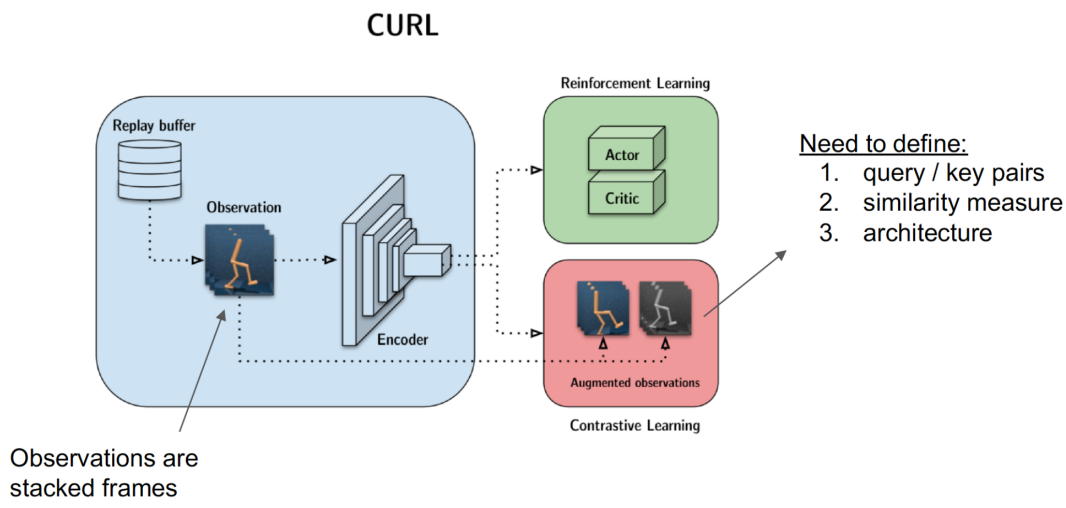
CovariantAI

The fact that RL worked so well actually inspired Pieter and his former students (Tianhao Zhang, Rocky Duan, and Peter Chen) to start a company called Covariant in 2017. Their goal is to bring these advances from the lab into the real world. An example is autonomous order picking.

3 - Unsupervised Reinforcement Learning

RL achieved mastery on many simulated domains. But we must ask the question: **How fast is the learning itself?** Tsividis et al., 2017 shows that a human can learn in about 15 minutes to perform better than Double DQN (a SOTA approach at the time of the study) learned after 115 hours.

How can we bridge this learning gap?
Based on the 2018 DeepMind Control Suite, pixel-based learning needs 50M more training steps than state-based learning to solve the same tasks. Maybe we can develop an unsupervised learning approach to turn pixel-level representations (which are not that informative) into a new representation that is much more similar to the underlying state.



CURL brings together contrastive learning and RL.

- In RL, there's typically a replay buffer where we store the past experiences. We load observations from there and feed them into an encoder neural network. The network has two heads: an actor to estimate the best action to take next and a critic to estimate how good that action would be.
- CURL adds an extra head at the bottom, which includes augmented observations, and does contrastive learning on that. Similar configurations of the robot are brought closer together, while different ones are separated.

The results confirm that CURL can match existing SOTA approaches that learn from states and from pixels. However, it struggles in hard environments, with insufficient labeled images being the root cause.

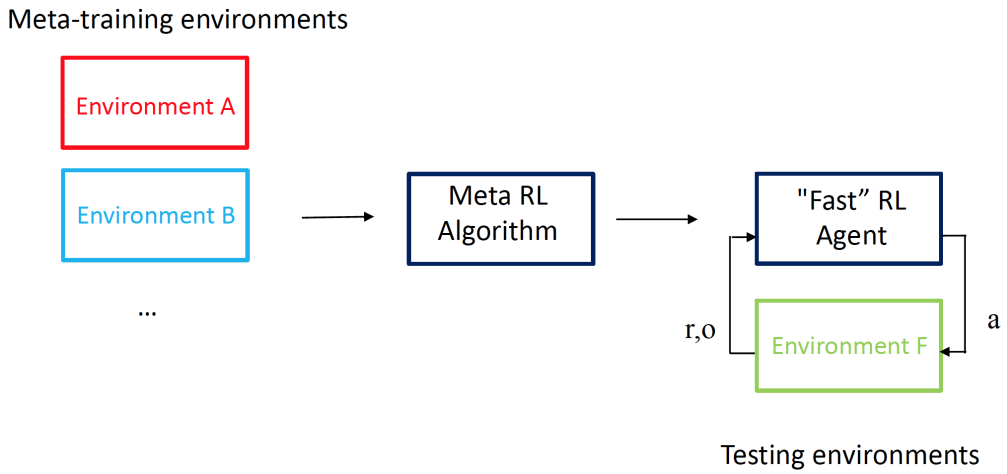
4 - Meta Reinforcement Learning

The majority of fully general RL algorithms work well for any environments that can be mathematically defined. However, environments encountered in the real world are a tiny subset of all environments that could be defined. Maybe the learning takes such a long time because the algorithms are too general. If they are a bit **more specialized** in things they will encounter, perhaps the learning is faster.

Can we develop a fast RL algorithm to take advantage of this?

In traditional RL research, human experts develop the RL algorithm. However, there are still no RL algorithms nearly as good as humans after many years. Can we learn a better RL algorithm? Or even learn a better entire agent?

RL^2



RL^2 (Duan et al., 2016) is a meta-RL framework proposed to tackle this issue:

- Imagine that we have multiple meta-training environments (A, B, and so on).
- We also have a meta-RL algorithm that learns the RL algorithm and outputs a "fast" RL agent (from having interacted with these environments).
- In the future, our agent will be in an environment F that is related to A, B, and so on.

Formally speaking, RL^2 maximizes the expected reward on the training Markov Decision Process (MDP) but can generalize to testing MDP. The RL agent is represented as a Recurrent Neural Network (RNN), a generic computation architecture where:

- Different weights in the RNN mean different RL algorithms and priors.
- Different activations in the RNN mean different current policies.
- The meta-trained objective can be optimized with an existing "slow" RL algorithm.
- The resulting RNN is ready to be dropped in a new environment.

RL^2 was evaluated on a classic **Multi-Armed Bandit** setting and performed better than provably (asymptotically) optimal RL algorithms invented by humans like Gittings Index, UCB1, and Thompson Sampling. Another task that RL^2 was evaluated on is **visual navigation**, where the agent explores a maze and finds a specified target as quickly as possible. Although this setting is maze-specific, we can scale up RL^2 to other large-scale games and robotic environments and use it to learn in a new environment quickly.

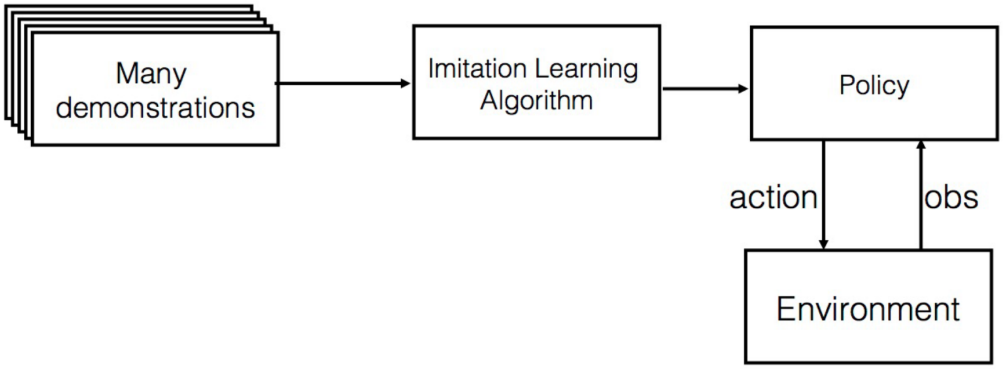
Learn More

- Schmidhuber. Evolutionary principles in self-referential learning. (1987)
- Wiering, Schmidhuber. Solving POMDPs with Levin search and EIRA. (1996)
- Schmidhuber, Zhao, Wiering. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. (MLJ 1997)

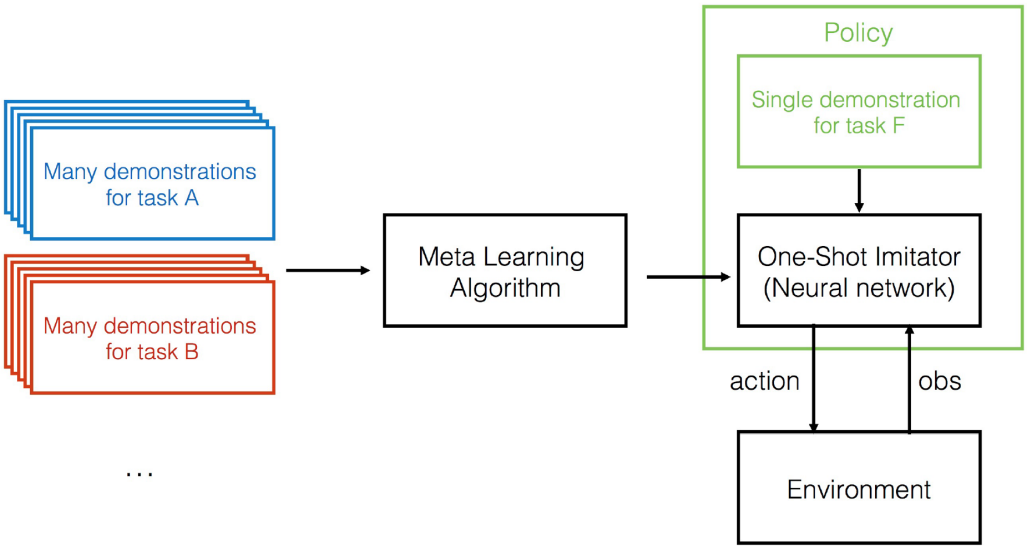
- Schmidhuber, Zhao, Schraudolph. Reinforcement learning with self-modifying policies (1998)
- Zhao, Schmidhuber. Solving a complex prisoner's dilemma with self-modifying policies. (1998)
- Schmidhuber. A general method for incremental self-improvement and multiagent learning. (1999)
- Singh, Lewis, Barto. Where do rewards come from? (2009)
- Singh, Lewis, Barto. Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective (2010)
- Niekum, Spector, Barto. Evolution of reward functions for reinforcement learning (2011)
- Wang et al., (2016). Learning to Reinforcement Learn
- Finn et al., (2017). Model-Agnostic Meta-Learning (MAML)
- Mishra, Rohinenjad et al., (2017). Simple Neural AttentIve Meta-Learner
- Frans et al., (2017). Meta-Learning Shared Hierarchies

5 - Few-Shot Imitation Learning

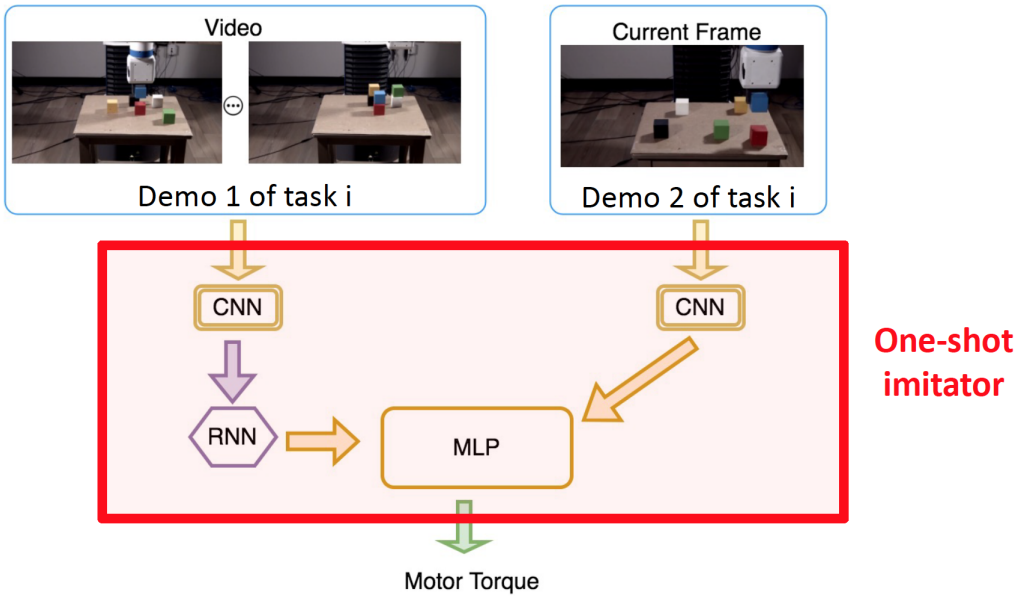
People often complement RL with **imitation learning**, which is basically supervised learning where the output is an action for an agent. This gives you more signal than traditional RL since for every input, you consistently have a corresponding output. As the diagram below shows, the imitation learning algorithm learns a policy in a supervised manner from many demonstrations and outputs the correct action based on the environment.



The challenge for imitation learning is **to collect enough demonstrations to train an algorithm**, which is time-consuming. To make the collection of demonstrations more efficient, we can apply multi-task meta-learning. Many demonstrations for different tasks can be learned by an algorithm, whose output is fed to a one-shot imitator that picks the correct action based on a single demonstration. This process is referred to as **one-shot imitation learning** (Duan et al., 2017), as displayed below.



Conveniently, one-shot imitators are trained using traditional network architectures. A combination of CNNs, RNNs, and MLPs perform the heavy visual processing to understand the relevant actions in training demos and recommend the right action for the current frame of an inference demo. One example of this in action is block stacking.



Learn More

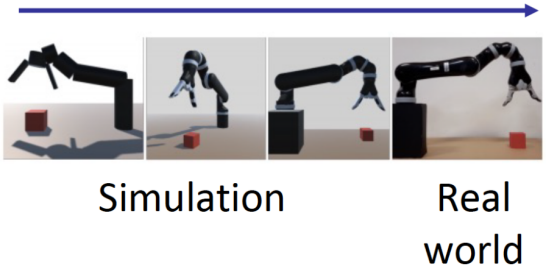
- Abbeel et al., (2008). Learning For Control From Multiple Demonstrations
- Kolter, Ng. The Stanford LittleDog: A Learning And Rapid Replanning Approach To Quadrupled Locomotion (2008)
- Ziebart et al., (2008). Maximum Entropy Inverse Reinforcement Learning
- Schulman et al., (2013). Motion Planning with Sequential Convex Optimization and Convex Collision Checking

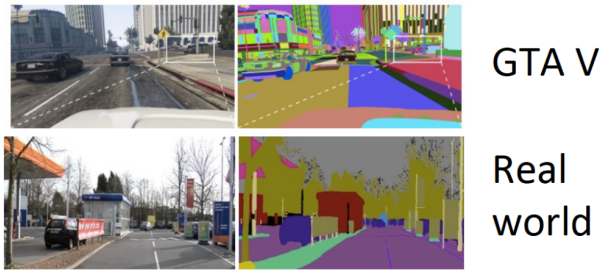
- Finn, Levine. Deep Visual Foresight for Planning Robot Motion (2016)

6 - Domain Randomization

Simulated data collection is a logical substitute for expensive real data collection. It is less expensive, more scalable, and less dangerous (e.g., in the case of robots) to capture at scale. Given this logic, *how can we make sure simulated data best matches real-world conditions?*

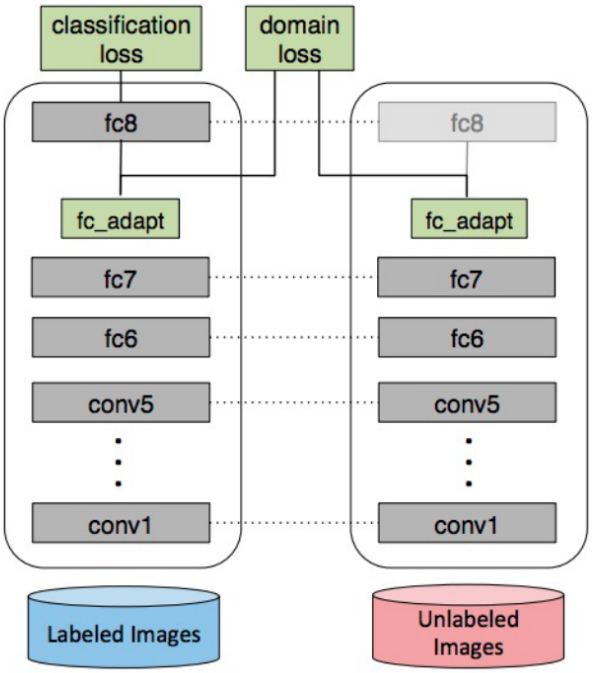
Use Realistic Simulated Data





One approach is to make the simulator you use for training models as realistic as possible. Two variants of doing this are **to carefully match the simulation to the world** (James and John, 2016; Johns, Leutenegger, and Division, 2016; Mahler et al., 2017; Koenemann et al., 2015) and **augment simulated data with real data** (Richter et al., 2016; Bousmalis et al., 2017). While this option is logically appealing, it can be hard and slow to do in practice.

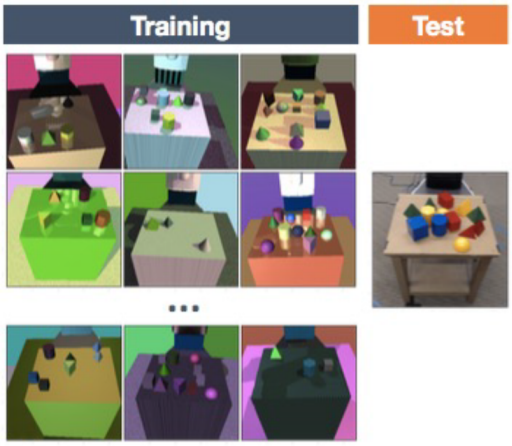
Domain Confusion



Another option is **domain confusion** (Tzeng et al., 2014; Rusu et al., 2016).

- In this approach, suppose you train a model on real and simulated data at the same time.
- After completing training, a discriminator network examines the original network at some layer to understand if the original network is learning something about the real world.
- If you can fool the discriminator with the output of the layer, the original network has completely integrated its understanding of real and simulated data.
- In effect, there is no difference between simulated and real data to the original network, and the layers following the examined layer can be trained fully on simulated data.

Domain Randomization



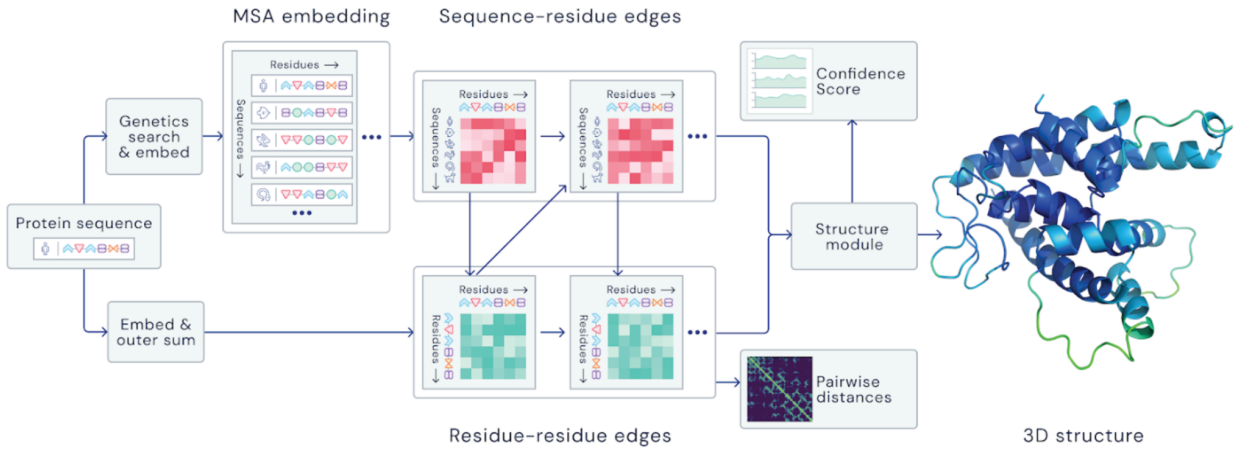
If the model sees enough simulated variation, the real world may look like just the next simulator

Finally, a simpler approach called **domain randomization** (Tobin et al., 2017; Sadeghi and Levine, 2016) has taken off of late. In this approach, rather than making simulated data fully realistic, the priority is to generate as much variation in the simulated data as possible. For example, in the below tabletop scenes, the dramatic variety of the scenes (e.g., background colors of green and purple) can help the model generalize well to the real world, even though the real world looks nothing like these scenes. This approach has shown promise in drone flight and pose estimation. The simple logic of more data leading to better performance in real-world settings is powerfully illustrated by domain randomization and obviates the need for existing variation methods like pre-training on ImageNet.

7 - Deep Learning For Science and Engineering

AlphaFold

In other areas of this lecture, we've been focusing on research areas of machine learning where humans already perform well (i.e., pose estimation or grasping). In science and engineering applications, we enter the realm of machine learning performing tasks humans cannot. The most famous result is AlphaFold, a Deepmind-created system that solved protein folding, an important biological challenge. In the CASP challenge, AlphaFold 2 far outpaced all other results in performance. AlphaFold is quite complicated, as it maps an input protein sequence to similar protein sequences and subsequently decides the folding structure based on the evolutionary history of complementary amino acids.



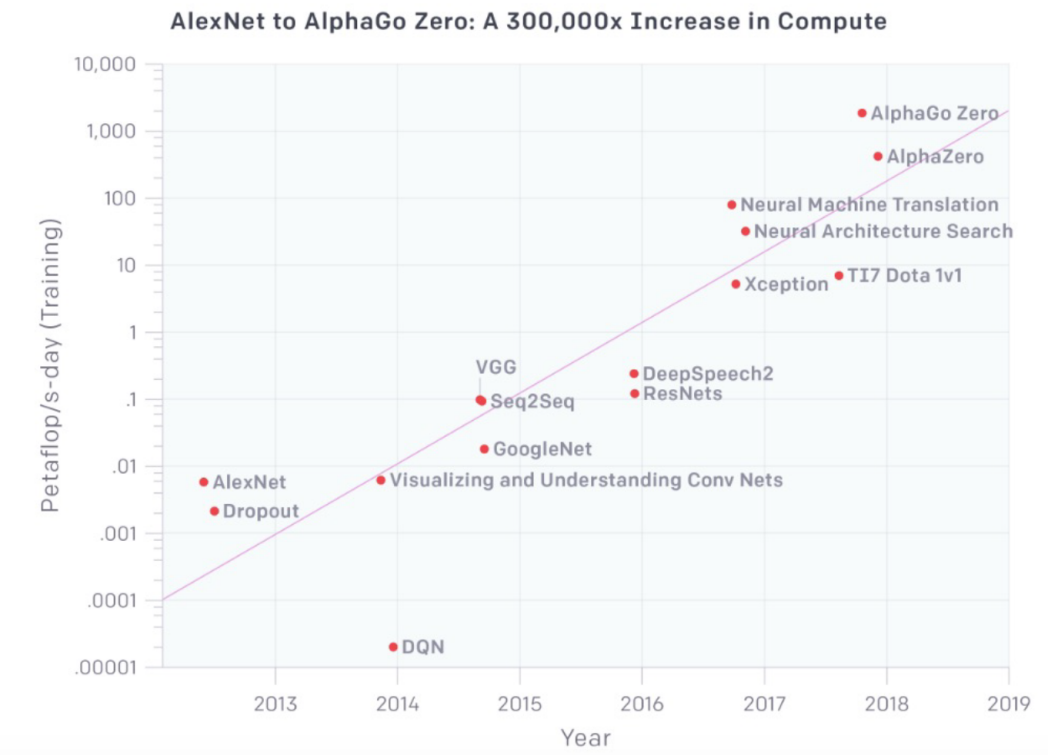
Other examples of DL systems solving science and engineering challenges are in circuit design, high-energy physics, and symbolic mathematics.

Learn More

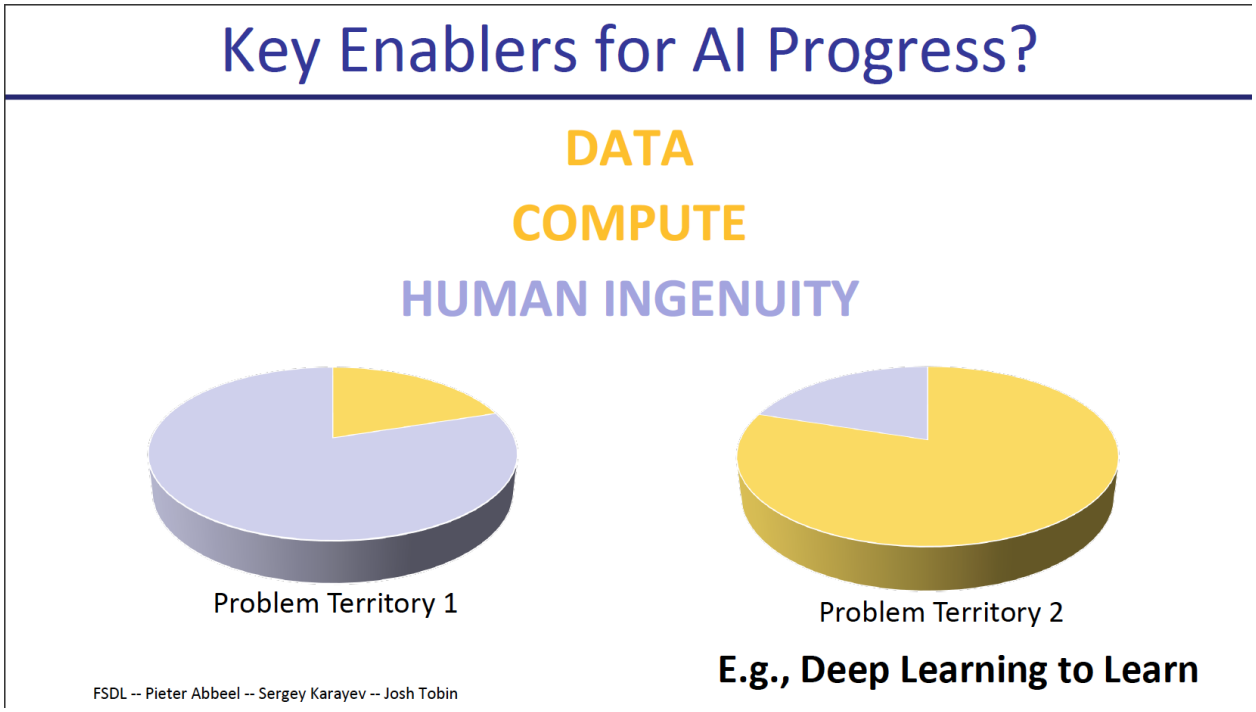
- AlphaFold: Improved protein structure prediction using potentials from deep learning. Deepmind (Senior et al.)
- BagNet: Berkeley Analog Generator with Layout Optimizer Boosted with Deep Neural Networks. K. Hakhamaneshi, N. Werblun, P. Abbeel, V. Stojanovic. IEEE/ACM International Conference on Computer-Aided Design (ICAD), Westminster, Colorado, November 2019.
- Evaluating Protein Transfer Learning with TAPE. R. Rao, N. Bhattacharya, N. Thomas, Y, Duan, X. Chen, J. Canny, P. Abbeel, Y. Song.
- Opening the black box: the anatomy of a deep learning atomistic potential. Justin Smith
- Exploring Machine Learning Applications to Enable Next-Generation Chemistry. Jennifer Wei (Google).
- GANs for HEP. Ben Nachman
- Deep Learning for Symbolic Mathematics. G. Lample and F. Charton.
- A Survey of Deep Learning for Scientific Discovery. Maithra Raghu, Eric Schmidt.

8 - Overarching Research Theme

As compute scales to support incredible numbers of FLOPs, more science and engineering challenges will be solved with deep learning systems. There has been exponential growth in the amount of compute used to generate the most impressive research results like GPT-3.



As compute and data become more available, we open a new problem territory that we can refer to as **deep learning to learn**. More specifically, throughout history, the constraint on solving problems has been human ingenuity. This is a particularly challenging realm to contribute novel results to because we're competing against the combined intellectual might available throughout history. Is our present ingenuity truly greater than that of others 20-30 years ago, let alone 200-300? Probably not. However, our ability to bring new tools like compute and data most certainly is. Therefore, spending as much time in this new problem territory, **where data and compute help solve problems**, is likely to generate exciting and novel results more frequently in the long run.



9 - How To Keep Up

"*Give a man a fish and you feed him for a day, teach a man to fish and you feed him for a lifetime*" (Lao Tzu)

Here are some tips on how to keep up with ML research:

- (Mostly) don't read (most) papers. There are just too many!
- When you do want to keep up, use the following:
    - Tutorials at conferences: these capture the essence of important concepts in a practical, distilled way
    - Graduate courses and seminars
    - Yannic Kilcher YouTube channel
    - Two Minutes Paper Channel
    - The Batch by Andrew Ng
    - Import AI by Jack Clark
- If you DO decide to read papers,
    - Follow a principled process for reading papers
    - Use Arxiv Sanity
    - Twitter
    - AI/DL Facebook Group
    - ML Subreddit
    - Start a reading group: read papers together with friends - either everyone reads then discusses, or one or two people read and give tutorials to others.

How to Read a Paper?

- Read the title, abstract, section headers, and figures
- Try and find slides or a video on the paper (these do not have to be by the authors).
- Read the introduction (Jennifer Widom)
  - What is the problem?
  - Why is it interesting and important?
  - Why is it hard? (e.g., why do naive approaches fail?)
  - Why hasn't it been solved before? (Or, what's wrong with previous proposed solutions? How does this paper differ?)
  - What are the key components of this approach and results? Any limitations?
- Skim the related work.
  - Is there any related work you are familiar with?
  - If so, how does this paper relate to those works?
- Skim the technical section.
  - Where are the novelties?
  - What are the assumptions?
- Read the experiments.
  - What questions are the experiments answering?
  - What questions are the experiments not answering?
  - What baselines do they compare against?
  - How strong are these baselines?
  - Is the experiment methodology sound?
  - Do the results support their claims?
- Read the conclusion/discussion.
- Read the technical section.
  - Read in "good faith" (i.e., assume the authors are correct)
  - Skip over confusing parts that don't seem fundamental
  - If any important part is confusing, see if the material is in class slides or prior papers
- Read the paper as you see fit.

[source: Gregory Kahn]

FSDL -- Pieter Abbeel -- Sergey Karayev -- Josh Tobin

Finally, **should you do a Ph.D. or not?**

- You don't have to do a Ph.D. to work in AI!
- However, if you REALLY want to become one of the world's experts in a topic you care about, then a Ph.D. is a technically deep and demanding path to get there. Crudely speaking, a Ph.D. enables you to develop new tools and techniques rather than using existing tools and techniques.