1 Introduction

There were two keynotes at CCN 2017 – Josh Tenenbaum said none of the AI systems we see are "real AI" – nowhere near what we observe in biology. There are some basic principles we have not figured out. This should be one of the main topics of research who are interested in pushing the science of intelligence. The paradigm we have for machine learning is not complete. Main of us know about supervised learning – you basically tell a machine what the output should be for an input. If it's wrong, you tell it it's wrong; if it's right you tell it it's right. This is like, say, 95% of the applications of machine learning. What deep learning brought to the table is the ability to train those systems end-to-end. Classically, you take the raw pixels and design a feature extractor, and then train a classifier. Deep learning has allowed us to build a cascade of trainable modules, all of which have tunable parameters, trained to optimize some performance measure which approximates the number of samples you get wrong. You can represent the input as a list of numbers, not necessarily a vector – lots of ways to represent inputs, including graphs – then you apply a linear operator to it, and a pointwise nonlinearity (ReLU). And then you keep doing this for some number of layers. You are forced during the process of learning to adjust the weights. This is a large-scale optimization problem. Everyone uses some form of stochastic gradient descent to solve this problem. This gives you a noisy estimate of the gradient. How do you compute the gradient? This can be done with backpropagation, which only uses the chain rule applied to matrices and vectors. This allows you to propagate signals backwards through the computational graph, which generally does not have loops (it can have loops, like in recurrent nets, in this case you unroll the loops). The people who invented this technique were not actually interested in machine learning – the idea that you could optimize a multi-stage system like this was essentially invented by people doing optimal control in the 1960s. People did not realize you could use this for machine learning until the 1970s, and made it work in the 1980s. If you want those machines to process images or text, or audio signals, you have to put some structure into the linear operators. You might need to add more types of operators than just matrices and non-linearities. This is where convolutional nets come in. You connect certain variables to certain variables, and parametrize in a certain way. So in an image, you have local correlations: you can extract information from an image by looking at a patch in the image, and encoding these patches with a small number of variables. This idea comes from neuroscience – Hubel and Wiesel's simple cells and complex cells. Fukushima 1982 was the first to try this kind of model. Backpropagation was not around yet really, so it did not quite work thta well. My contribution a few years later was to simplify this architecture a lot and train it using backprop. You can see that the representations at each layer are shift-invariant. It was a very important fact that you could recognize multiple objects without having to separate them from each other. At the time, the wisdom was you have to separate objects first before trying to classify. This ended up going in banks around 1995 for digit recognition.

After 1995, interest in neural nets kind of died. At the time, it was difficult to implement things. No Matlab, no Python – you had to implement your own interpreter. Now we have

Tensorflow, Pytorch, etc. So most people stopped being interested in neural nets. I actually did not really work on machine learning between 1995 and 2001. I started again at NYU. One of the first project I did was: Can we use machine learning to drive robots? (DARPA). We trained a convolutional net to predict where objects were to avoid them. So they saw it work, and funded a bigger project called LAGR. This had the ability to quickly recognize obstacles and change direction; we also needed to combine it with control. Soon we realized semantic segmentation was also important. Around 2009, datasets started appearing for this: We had a small number of samples, so we trained small convolutional neural nets. We implemented the whole system on an FPGA which could run at 20 frames per second. So we demonstrated you could do real time vision quickly.

In 2013, Ilya Sutskever and Alex Krizhevsky were able to beat ImageNet record by a huge margin after implementing convolutional net on a GPU. Using bigger and bigger neural nets, people were able to beat even humans (currently 3% error rate). Some of the main ones over the past few years: VGG, GoogleNet, ResNet, DenseNet. ResNet has skipping connections, it allows you to add a lot more layers since it avoids the problems in optimization of weights dying. People on Facebook upload about 1 - 1.5 billion photos every day. Every photo goes through four convolutional nets. One recognizes events, one filters objectionable content, one produces captions for the visually impaired, and the last one does face recognition (activated on certain countries or continenets). Why so many layers? There are two possible reasons: 1) a lot of functions we need to compute are essentially "sequential" – reduce any function to two steps. CNF and DNF, you can compute any boolean function. You can approximate function as close as you want using some high dimensional basis functions. You can compute anything with two layers. For years, theorists would ask why is it useful? You can approximate anything with two layers. But this might not be efficient. So what is it with many layers is a good idea? It is probably because the world is compositional. We can recognize images and objects – first edge detectors, then assemble those edges into motifs like corners, then assemble those into bigger motifs, and then into objects, and then into scenes. This works because the world is compositional. It is interesting that the world is described by "simple functions". We can now train on ImageNet in one hour by parallelizing on 256 GPUs. But no one has a satisfying solution to the problem of training in parallel efficiently. The efficiency goes down as you increase the parallelization. What distributed optimization algorithms are good to make this efficient?

MASK R-CNN draws outline for each object. This is useful for self-driving car systems. In our brains, in visual cortex, the ability to identify objects and locate objects talk to each other (ventral and dorsal) – here there is a bit of that, but mostly done in a single system. The performance is pretty astonishing. It is possible to run 5 times per second on a smartphone, though it drains your batteries. It does this real-time, and also can detect poses for humans. Very recently, there is semantic segmentation for 3D objects (ShapeNet). The task is to identify 3D objects and segment them. The winner of the most recent competition is Submanifold Sparse ConvNets. You can make a 3D tensor of the voxels. Most voxels are empty, it's a waste of time to do this on empty voxels. So they only do convolution on places where you have non-empty voxels. That is all done efficiently and is run on GPU and

everything. You can also define convolutions on regular graphs. The connections indicate the fact that neighboring pixels are correlated. The convolution is a local operator there, it can be defined in Fourier space. It becomes a diagonal operator (elementwise multiplication), this lets you define for irregular graphs. You can compute the Graph Laplacian, diagonalize, and the rotation into the eigenspace is the Fourier transform. In this space, doing pointwise multiplication is the convolution. This is a cool new avenue, there will be a tutorial at NIPS.

We also have a system using gated convolutional nets, where you feed a sentence to be translated as a sequence of vectors – this all goes to an attention module which tries to match inputs and outputs. This has the advantage over recurrent nets in being fast and parallel, and can handle sentences being very long. There are a whole bunch of projects at FAIR. Research should be open: make the field progress faster. There are techniques we do not have which we need the whole community to solve. See https://github.com/facebookresearch. We mostly use Pytorch. There are other applications of convnets: Medical imaging, in science, physics, etc, games, and so on.

Deep learning kind of breaks basic rules of statistics, which is why people are skeptical at first. The models are huge, there is no explicit regularization, nonconvex, no guarantee of convergence, little attention is paid to managing uncertainty, and a lot of effort is spent on computational issues. It goes against common wisdom. There is something we don't understand that goes on here at the theoretical level.

2 Obstacles to AI

Animals and humans learn considerably more efficiently than other machines. So what is the learning algorithm of the brain? Maybe there is some unique principle. What is the equivalent of aerodynamic intelligence (birds vs. planes, but same principle makes them fly)? How much prior structure does animal learning require? This is the old nature/nurture debate. A lot of animals and humans have "common sense". In my opinion, it is the ability to fill in the blanks. You get a partial view of the world, and you are able to fill in the blanks. You can fill in the view in if you only see half my face. You can extrapolate for occlusions. In some sense, the essence of intelligence is to predict the future. Ability to predict the blanks from partial descriptions. If we had a learning paradigm where the machine has to predict whatever it does not know, what your world is going to look like, is "predictive learning". Orangutan sees a cup that's full, is then shows it's empty – it laughs because its model of the world has been broken. Compare to a baby playing peekaboo – no notion of object permanence. Also if your model is broken, it can be surprising – you have to pay attention since there is something to learn, or it is scary. Emmanuel Dupoux has some research on this in psychology. So what is missing from our learning paradigms? We have reinforcement learning, supervised learning, unsupervised learning. Unsupervised learning is the idea of predicting everything from everything else – try to predict everything in the future, and check your prediction against what is actually there. Here the amount of info the world gives to the machine is enormous, you have to output detailed representations of the future. So there it is very complex, you give it a lot of feedback and information. Supervised

learning you give presumably less feedback – you give it less feedback, you can only give it for datasets which are labelled by humans. Animals and humans do not do it this way, they learn by observation. They are not given labels at all. Orangutans are solitary animals. In RL, there is weak feedback – you wait for the machine to produce an answer, you just saw whether it is good or not. That is not enough to get a machine to learn anything without a ton of trials. This relates to the cake analogy.

The RL case, if this is the only mode of learning, the number of trials you have to do is very large. Obviously we are not doing this when we learn to drive. Pure model-free RL requires many trials to learn anything, which is ok in a game, not ok in the real world (anything you can do in the real world can kill you). So you are liable to try things in your head in advance and plan. The other nice thing about trying things in your head is that it is faster than running things in the real world (in real world, you cannot run things faster than real time). RL works great for games because you can simulate the environment in a computer – you can play more games than all of humanity in the last years; it is an incredible engineering achievement. Does it apply to AI? Unclear. We also work on this at FAIR – Doom, StarCraft.

Here is a quote from Rich Sutton, RL grandfather. He was proposing architecture Dyna: The idea is trying things in your head. This suggests a primitive process – this is the more primitive notion. Classic model based optimal control has a plant, it tells you the state, and the command, and you can roll out the sequence of actions via backprop through time and gradient descent, you can figure out sequence of commands that will get the optimal trajectory. This is what Nassa has been doing for a long time. This is known as the adjoint state method. We have to get machines to learn models of the world via observation. So the next revolution will not be supervised.

So let us have a complete AI system: you have an agent and an objective. The agent is trying to get to a state where the objective function is happy. If the agent is to be intelligent, the agent has to have some sort of model of the world which allows it to predict consequences of its action and learn to plan. It is possible when you have robots, you can write down dynamics – handcrafted and identified, then you can do planning with corrections for things that are unexpected. But what if you get into a room and have to figure out the situation? That is a problem we have. You have to give action sequences to predict a function. People in RL call this rollout: you initialize a system with current state of world, run world simulator for a proposed action sequence, and then update the model of the world and the proposed action sequence. How to learn models of the world? In the past, we have a game engine – this makes fuzzy predictions, it cannot predict exactly what will happen. There is also a lot of work on language with dialog systems. If you use predictive systems, they learn faster than if you don't.

What is unsupervised learning, or predictive learning? Your world consists of two varialbes, one pixel each. These are observations for the world, (y_1, y_2). How do I represent the knowledge that the world lies on this curve (parabolic). You can train a contrast function, high on things you do not observe, and low on things you do observe. The problem is, what about points you do not see? How do you make their energy high?

Adversarial training is the idea that to generate points which are outside, you get a neural net to train these points. You can interpret this backwards: Let us say you want to build a box which observes past and predicts future, and you have some random vectors. Let us say you put a pen on a table and ask it where it will go? You ask the system. You can think of set of plausible futures as some sort of manifold – your prediction is here, its prediction is there, but you do not want to penalize it if it is still on the manifold of possible futures. So you have a generator which produces proposals, and a discriminator which trains itself to recognize stuff which comes from the generator manifold (learns a contrast function). As you train these things together, the discriminator starts taking shape, and then you get good predictions. These things work amazingly well. You can do face arithmetic, generate images from Imagenet. There is a whole menagerie of types of GANs – for instance, Creative Adversarial Networks. They trained a network on various styles. Trained on 75000 paintings, rated highest by human subjects.

Training with least square you get a fuzzy prediction, because it cannot decide which future will happen - you get an average. With GANs you get a sharper image, because it instatiates a specific prediction.

Video prediction: predicting five frames pixels. You can also predict semantic segmentations. What happens here is when pedestrians stop crossing the street, they keep crossing the street. You can imagine this would be very useful for self-driving cars.

The last thing I will show is a very recent paper: The idea here is to predict the default average prediction (multiple things can happen), look at the past, run througha neural net, predict the future, compare with the future (no good because of multiple futures depending on what the world decides to do). So then look at the difference between the vectors, and then figure out how to train parametrized function to learn a latent variable which when added to the model will correct the mistake. This is a dataset by Levine at Berkeley. You poke an object and get to see where it is after the result. When you change the action (the random vector) it figures out the object can move in different locations depending on what happens in the world.

3 Questions

(a) A part from all these things like detecting faces etc., do you think artificial intelligence can solve social issues? Answer: Yeah. The example of something in the press, if you do not use machine learning properly, this results in biased systems which affect the bias of data which is trained on. There are famous failures. There is research that exists for de-biasing data to let machines give decisions which are less biased. If anything we need more AI not less if we want to reduce bias.

(b) Do you think predictive model for predicting the world can improve RL? Answer: There were papers in the 90s about RL saying that model-based RL did not have convergence proofs. It is harder to apply model-based RL versus model-free. If the model is not completely accurate, there is an irreducible amount of error. People have

not figured this out yet. The idea is you are learning model of world by observation and interaction, with RL you can map representations of the world to actions which are useful.

(c) For bedrooms it is easy to evaluate whether an image is a bedroom. Can you talk about evaluating generative models? Answer: There are ways, but people do not agree. There is Inception score, you hope that it is not biased, but it is biased because it is trained on ImageNet. There are lots of situations like this in ML. Translation for instance. Optimize BLEU score doesn't provide any worse of a model. Problem of good objective functions and measuring performance of generative models is hard.

(d) What about the theory? Answer: There is a thing I am organizing at UCLA on this topic. There is a lot of theory going on. First, what is the geometry of the objective functions. One thing which seems to be emerging is that getting trapped in local minima is not really the case in a huge amount of dimensions. There are lots of ways to escape. There is other evidence: You start from random initial conditions: you get same performance, but not same solution. If you start from two cases, get two distributions, is there a straight line from one to the other? Can I bend the trajectory to go from one to the other? Yes. There is an entire subspace of degenerate solutions which are well-connected. There are theoretical results which attempt to show this. Random matrix theory: the idea that for spin-glasses, can show the distribution of local minima, low-energy, most minima are in a small range, that is probably what SGD gets. There is a paper from UCLA on how the randomness of SGD is very efficient. Why does it work well despite fact of a huge number of dimensions? What is the justification for convolutional architectures, are there first-principled ways of deriving this? Yes: random process shaped by noise, characterize the filter. The voice is a noise excitation, your voice is shaped by vocal tract. If you have a single sound, you can detect the entire signal, pass a filter bank, rectify, and average the output, you can get the filter used to predict the signal. That is basically the first few layers of convolutional net. Unfortunately, speech signal changes over time: You can do this over time though. This is convolution on small windows. Unfortunately some noise goes through. So you have to repeat this process. This is a paper by Max Taggert.

(e) Can a real AI understand the world like a human? Answer: Very different ways.

(f) Is anyone doing causal research at FAIR? Answer: There are a lot trying to figure out this stuff. It is very complex. At Facebook it's Leon Bottou.

4 Takeaways