1 Introduction

I have been doing machine learning for a very long time, this has been around for a long time, and a machine technology since the sixties in Russia. But it's always been complicated to get a leg in. But, it is an end-to-end process: You have to understand how data is acquired, normalized (turn data into numbers), which numbers are useful, and then at that point and only that point do you run your algorithm. That's the way things used to be. For instance, for text, you'd do bag of words, TF-IDF, LDA, CRF or something. So before for machine learning, you had to understand every part of these pipelines, each of which is complicated in itself. For instance SIFT/HOG in computer vision. Building blocks of SIFT/HOG are essentially those of convolutional networks! 5 years ago, in image processing, there was a major advance: 10% jump on ImageNet dataset (now we've overfit it to roughly 0% accuracy).

I think it is inarguable that deep learning has been a successful and mature technology: image detection, real time translation on your phone. A lot of this can enrich humanity, superhuman performance on Go, robotics systems, but the thing where we have to pause in machine learning: As we push the boundaries of these things, and they start ot interact with people, the bar for working changes. Doesn't mean just 100% classification on a dataset: Cars are scary. The training set for Mobileye system in Tesla wasn't able to detect a reflection of a truck which took a turn. This caused a crash. We have to be sure they work, and we have to understand how they work. This is where theory has a place: ensure safety, trust, predictable. So how can we bring that kind of understanding to machine learning: Understand when it's good and bad.

2 How does a theoretician approach deep learing?

Well, multilayer perceptron has been around for a while. How does this work? You get an input z, and output $\phi(z; x)$. Then you compute the loss function, plug it in, and some deep learning library gets you an answer. How does this work? I'm in optimization. It looks messy, everything interacts in nonlinear way. What makes this optimization hard? Now we're getting into stuff from the 90s. How does nonconvex optimization work? I personally think 1D pictures are misleading (all this spin-glass model type of stuff). The literature is unfortunately not clarifying: three papers which say completely different things. It's not clear to anybody what makes the optimization work inside of deep learning. Let's not dwell on these contradictions. Well, do we actually believe that optimization is hard? We never actually run this thing until it terminates. If you download standard code and turn off all the regularization mechanisms, you can get it to 0. Deep networks are globally easy to minimize on MNIST. It also happens on CIFAR-10. Lots of other datasets too. So now we have a weird issue: optimization is not the hard part of deep learning. Optimization is easy. The hard part is what happens out of sample: Error starts creeping up.

Now, when you use Alex Krizhevsky's parameters (intricate), the train/test set error track each other. Now, you're NOT supposed to look at that out-of-sample performance,

but I guarantee you he looked at it. So we're minimizing a loss (usually something we invent). This is a softmax multinomial regression loss – and we care how many images are we giving wrong assignment to – eventually, the line (for classification) flattens out. When we overfit to in-sample, the classification error does NOT go up. So the issue is not optimization, but it's generalization. Generalization is all we really care about. That's what all machine learning is about. For me, the classic thing here is you have data, will use the data, and all we care about is how we do on new data.

So if you get i.i.d. samples from some distribution $\mathcal{D}$, you want ot find a good preditor function. If you have new sample z_{n+1}, want to do well too. So we would like to minimize with respect to expected loss. We want $\mathbb{E}_z[\text{loss}(f, z)]$ to be small. Too issues: don't know distribution, integrals are hard. Instead we rely on law of large numbers: The sample average should be close to the expectation if n is large enough. We minimize this using stochastic gradient descent. Two issues: 1) is it close, 2) is SGD the right algorithm? So we care about generalization error. We want that to be small. We don't necessarily care about the value, we just want to not be surprised. We can't understand when it is a good proxy for the true population. So we need some tests for that.

Theorem 2.1. *Fundamental Theorem of Machine Learning.*
$R[f] = (R[f] - R_S[f]) + R_S[f]$ *The population risk is the generalization error plus the training error.*

We just want generalization error to be small. Zero training error does NOT imply overfitting. The second version: $R[f] = (R[f] - R[f_{\mathcal{H}}]) + (R[f_{\mathcal{H}}] - R[f^*]) + R[f^*]$, where f^* is the best in class. This is error vs. best in class (error vs. best in class, approximation error, error of best). But you can't calculate any of these. This however motivates the bias-variance tradeoff. But deep learning has shown us that that is completely wrong. I think we knew this already but didn't propagate it down to the undergrad classes.

3 How to reduce generalization error?

In deep learning, models where $p > 20n$ are common. You can make it super big.

How do we reduce generalization error? We've given up on model capacity with deep nets. Then "regularization": this to me seems like "just make the optimization bad". Another one is data augmentation: generate fake data: This essentially imposes on the algorithm what symmetries the data obeys (by sticking it in human wise). So for instance, reflections on image data. Or crops. This is a very powerful idea. Deep learning scales linearly with number of data points, which allows it to take advantage of this kind of augmentation. All of these things are sufficient but by no means necessary.

At Google, you have unbounded compute and can explore parts of deep learning that you can't as an academic. So we looked at this dumb dataset, CIFAR-10. People read much too much into performance here, but it's interesting. These are not really having anything to do w/image classification, but whatever, it's a benchmark.

So what happens if we turn off all regularizers. We downloaded CudaConvNet: 145578 parameters, with only 50000 datapoints. You get some non-zero training loss, and error goes down to 18%. There is a lot of tuning to make regularization work. Note that $p/n = 2.9$.

So you can tune parameters, OR, you get a bigger network. MicroInception has $1,649,402$ features, $p/n = 33$. Train loss is 0, train error is 14%. Then ResNet, the hot thing these days, make $2,401,440$ parameters, then $p/n = 48$, same story, reduce error. Now this is strange, making parameters bigger seems to reduce generalization error.

Now another weird thing: Even with random labels, you get training error to zero (with terrible generalization error). You can do other weird tests: shuffle pixels (in the same way) for all images. You can still get training error to zero. Now test error is 50% error instead of 14% error. Shuffling different portions of the data gives you linearly increasing test error. But you can still get training error to zero. If you feed in Gaussian white noise, can still get training error to zero. So they have ridiculous capacity, but when giving it real data, they generalize.

So the point here is that it's not just the model – that is not what contains the extra information. It's the model and the dataset together which generalize. Well, people think CIFAR-10 is a toy. What if we do image net? here, $n = 1.3M$ examples, $d = 150528$ (resized to $256x256$, $k = 1000$ classes. On ImageNet, we used actual Inception (27 million params, $p > 20n$). Here we used the standard Google regularizations. Training error would be 13.7%, Test is 23.4% accuracy. This is actually pretty good. Now if you turn off regularization, you lose a few percent of top 1-accuracy, but it's not too catastrophic. If you turn off both fake data and regularization, it's bad. I feel like fake data is what you're supposed to pay attention to.

Now using random labels, you get 5% error training data. I'm confident if we tuned things, we could get 0% error (maybe with 10 GPU years). So on real datasets, you can have these architectures get low training data and fit on random labels.

So now the question: Are neural nets learning or memorizing? I thing a little bit of both. It used to be in Google Translate, that "you abuse our patience" is translated to a line from Cicero. So it's clear that they are memorizing things. As soon as you reveal a mistake, they fix it behind the scenes (manually).

4 Recht Linearization Principle

Now, if we can't understand what machine learning algorithm does in case of linear regression, we have no hope of what it does in deep learning case. "Would you believe someone had a good SAT solver if it couldn't solve 2-SAT"?

So, why do we generalize when fitting the labels exactly? Well this totally happens in linear models. If you run SGD, you find minimum ℓ_2 norm solution (consider least squares). Least squares actually works quite well for classification too. There are infinite number of global minimzers, but we find the minimum norm one with SGD. I actually spent 10 years of my life thinking about problems like this. i.e. Compressed sensing. So the point is you have to pick some solution. Maybe in neural nets we pick one by messing with architecture. But

there are already pretty sensible things about sparsity and smoothness that already work. L'ets first show that things really do work for linear regression. There's this old thing called kernels, so you'd build a matrix with all your data (you never need more data than number of parameters). Now these datasets that seemed large in the 90s, youc an solve in about 3 min. So what happens if we do this on MNIST. You get 1.2% test error. Now if you add a wavelet transform, it goes to 0.6% error. So indistinguishable on the unit test.

So what about CIFAR-10: well, we get 46% error without pre-processing. If we do a random wavelet transform (1 layer convet), then you get 16% error, + ℓ_2 regularization gets this to 14%. But then solving linear systems scales as a factor of 2.7 or so, so it's much harder to use data augmentation.

5 Margins

As long as deep net doesn't do anything weird in the middle, as long as the representations are far apart from each other, things are nice. Data is nice, that's great! We can do well if there's a margin. So what exactly is going on in deep learning? When we minimize the norm, the inverse of the norm is the margin. So the smaller the norm, the bigger the margin is on your data. So if you can find things with large margin, we will generalize well. The test error is basically equal to the norm divided by $\sqrt{n}$. For kernels and stuff, this is a solved problem.

Challenge: Find comparable, reasonable margin bounds for deep learning that explain experimental phenomenon. There's a paper by Matus Telgarsky, Peter Bartlett that people do this. I think this can explain why deep learning can actually work. It's not just the model, it's about the data.

So this Recht linearization principle is a fruitful research direction: Recurrent neural networks (Hardt, Ma, R. 2016). Generalization and Margin in Neural Nets (Zhang 2017). Residual Networks (Hardt and Ma 2017). Bayesian Optimization (Jamieson 2017), Adaptive gradient, reinforcment learning as well.

6 What can deep learning learn from linear regression?

So it's a lot easier to get in to machine learning. Just because you get something that can work, doesn't mean you should ship it (lesson for all software). Andrew Ng talks about how artificial intelligence is the new electricy. I would say "Artificial Intelligence is the New Alchemy". It kind of worked though! It was the predecessor to chemistry (they invented gold plating). They just had no idea why it worked. Chris Wiggins: thermodynamics before Carnot. Or electricity before Maxwell's equations. Maxwell had no idea that electromagnetic waves existed, but his equations predicted it.

On the other hand, I have no idea what convolution filters do – adversarial examples. We are builidng tools for automatic fact checking and place ads, etc. We can scale to billions of people, but we don't undersatnd and when it breaks bad things happen. Elon Musk goes

around saying outrageous things. The terminator is not coming to get us. There are serious problems coming to get us threatening our democracy, and this guy is wasting time building bots for video games. There are unforseen consequences, we need to understand why it works.

Should not replacing mathematical modeling with robots. We shouldn't necessarily even replace humans following clear rules with robots unless we understand.

The positive note is the following: we've built airplanes, power grid, information infrastructure, Amazon, etc. Lots of amazing success stories. Machine learning should be one and can be one, but we need to add robustness and introspection to these models. This is the challenge for the next decade.

The margin view kind of renders the adversarial example view sort of trivial. But small perturbations in ℓ_∞ are large in ℓ_2. There's a paper from 2002: SVM for gender detection. Support vectors look androgynous.

I think image manifold view is misleading too. Even in linear space you can leave manifold where there is no data. Important to understand that there is something about process of generating the data that we understand. Fun thing about data driven modeling is learning from it and learning how to do without it.

7 Takeaways

Deep nets are powerful model classes. How is it that they can get zero error on training data (even if you permute the labels, etc.) and still generalize? Let's not forget about the importance of the data on generalization. If the data, in some space, has a large margin, then it should be able to generalize well. Can view deep learning as finding a representation which has large margin. We don't care what weird stuff it may do away from the margin, we won't lose anything perturbation wise for those examples. Only examples close to the margin are at risk (why classification error may not rise even as surrogate loss rises). Notably, this (to some extent) explains adversarial examples: Just construct "images" close to the margin (?). Other reasons why it's powerful: most powerful kind of "regularization" may have to do with the data input: It's possible to encode the structure of the data via data-augmentation strategies (like in images, for example). Deep learning wins and can take advantage here because it is a super large model class with lots of expressive power for which TRAINING TIME IS LINEAR IN THE NUMBER OF SAMPLES via SGD. This is possibly why it wins out over kernel methods, which require solving linear systems $\mathcal{O}(n^{2.7})$. (If that could be improved...)

8 My Questions

1. Adversarial example question: Is the claim that adversarial examples exist because it is easy to construct inputs close to the margin? (and thus it's trivial they will exist?)

2. Power of deep nets from the data: I agree that deep nets have had their biggest success on images. In fact to me it seems like almost all of the deep net successes are focused on images. What properties of images do you think are the cause for this success? What properties of the data do you think are most important to have for deep nets to work well? In some sense, it feels to me like images (and perhaps speech, other natural signals) have a lot of inherent symmetry built in and feel, to some extent, "additive". Am I getting it right when I say that this is to some extent what you mean by paying attention to the actual data? What is your opinion about deep nets on natural language type tasks?

3. Do you believe that if kernels could be trained in a manner s.t. time complexity was linear in the amount of new data they could reach a similar performance to deep nets, from learning theory perspective? Alternatively, if you could encode the invariances of the data in a small number of examples, performance of kernels should be as good, right?

4. How do you feel about random featurizations of data w/r/t deep nets?

5. Thoughts on algorithms to construct datasets?