Mathematics of Deep Learning Scribe: Kiran Vodrahalli
Princeton University 03/20/2018

1 Toward Theoretical Understanding of Deep Learning (Sanjeev Arora)

1.1 Introduction: Mysteries

Here are some mysteries. Why/how does optimization find globally good solutions to the deep learning optimization problem? We know it goes to a good global solution since the training error generally goes to zero – so we know it's finding good solutions.

Why do these nets generalize/ predict well on unsenn data? For example, VGG19 on CIFAR10: It has more than 6M variables, and it gets trained on 50K examples. Why does it do well? That seems to overturn insights from classical statistics.

What are nodes expressing? How is depth useful? How interpretable are the learned nets?

The explanations will probably overlap.

1.2 Understanding Nonconvex Optimization

If you look in the literature, there are lots of papers in the literature where people analyze nonconvex optimization for a lot of problems, like matrix completion, topic modeling, sparse coding, tensor decomposition, HMM learning, and so on. All of these results need to make assumptions about how the data was generated. Some assume a probabilistic model – and the optimization landscape is understood, the optimum is the ground-truth model which generated the data. So the main idea behind the analysis of these algorithms is to use this idea. You prove that the direction of movement at any point θ is postively substantially correlated with the desired direction $(\theta - \theta*)$, where θ^* is the global optimum. This leads to global convergence. If you had full foresight, you'd want to move exactly on that line. Instead you're moving along the gradient, with local improvements. As long as it's positively correlated, it'll make progress. You can see the framework for analyzing nonconvex optimization in offconvex.org. This can only work if you understand something about the landscape.

Now the problem in deep learning is that the optimization landscape is unknown - but wait, you have the loss function! But we lack a clean mathematical description of the inputs x_i and the outputs y_i: What describes an image of a dog? So maybe you have a function with random or unknown coefficients. But you don't have any idea of a landscape. Given that we lack this mathematical description, we're stuck with analyzing nonconvex optimization in an *unknown* landscape - so you can't really hope to have any kind of provable guarantee. So what are people doing? They're analyzing gradient descent in a nonconvex landscape, and people are applying ideas of optimization from long ago to show local optima. There are then conjectures that local optima are good enough.

Now suppose that the gradient is not zero. Then there exists some descent direction, and you can reduce the loss. But if the 2nd derivative is high, this allows the gradient to shift a lot! And the direction you thought was a descent direction may not be a descent direction.

So to ensure descent, you have to take small enough steps determined by the smoothness of the function. Once you trade those off, you can conclude that you're guaranteed to get close to a point with gradient close to 0, with speed determined by smoothness and the Lipschitz constant.

Is this enough? No, because there could be saddle points. A saddle point is where you could be a minimum in $d-1$ dimensions, but a maximum in the last dimension. But how do you find that direction in high dimensions? A nice paper suggests you add noise to gradient descent. This is called perturbed gradient descent. This can be analyzed like a random walk and show that within poly(d/ϵ) time, you can get to an ϵ-approximate 2^{nd} order local minimum. This analysis has been improved to get a near optimal result. Physicists are interested in Langevin dynamics, which is also a form a perturbed gradient descent with large noise.

Is there a recipe for optimization? Perturbed gradient descent actually isn't the recommended thing by these folks. People think that stochastic gradient descent kind of acts like this already.

Also from statistical physics are models based on spin glasses, and landscapes based on that – they suggest that many local minima are almost as good as a global minima. There are lots of tools here.

1.3 Expressiveness and Depth

What's the role of depth? One ideal result would be to show that depth helps. In principle, you could do all of deep learning with depth 2, but this would require very large size deep nets. The ideal result would be to show that for natural learning problems, you can't do it with depth 3, but you can do it with depth 5 or whatever. Now this is not within the reach of theory because we lack mathematical formalization for what a natural learning problem is, like differentiating cats and dogs. There are results like this for less natural problems.

Actually, there are lots of things like this in computational complexity, but the problem definitions are not so natural.

We have a recent paper with an observation that so far as I can tell is new – adding depth can act as an acceleration for optimization. Deep learning has been around for a long time, but it never really got going because training deeper nets is difficult. The feeling was that gradient gets attenuated through layers, and you don't have a clear direction for improvement. The feeling was that adding depth is a hassle. Here we are suggesting that adding more depth can act as an acceleration (technical word) for optimization. There's a technique called Nesterov momentum. The idea is that you don't just do gradient descent, but also keep track of past gradients. The direction of movement is then a function of past gradients. For convex optimization, one can show that you can accelerate the optimization (improve rate of convergence). One example is ℓ_p regression, for $p \geq 2$. Even this you can accelerate by adding depth. Now we're going to change this classical convex problem. Here we just apply a simple overparametrization: Replace the vector w with a vector times a scalar w_2: $L(w_1, w_2) = \mathbb{E}_{(x,y)\sim D}\left[\frac{1}{p}(x^T w_1 * w_2 - y)^p\right]$, where w_2 is a scalar and w_1 is a vector. Now the optimization landscape is very different! But this landscape has acceleration terms, and it's kind of like acceleration. This also happens in practice. It beats standard acceleration methods, and it beats these methods out of the box. That's just a teaser for a more detailed talk by Nadav Cohen.

1.4 Generalization Mystery

Why does a deep net classify unseen data well? Training loss goes to zero typically (that's actually a mischaracterization, cross-entropy loss can be close to zero with still things happening), and testing error also gets close to training error! There have been lots of attempts (information bottleneck etc.) but these are more suggestions, you cannot calculate from these theories.

There was a paper by Chang et al 2017: Deep nets can fit random labels - so they have capacity. For some time I used to think ok, maybe it's just an academic mystery - you relate it to VC dimension and all these things. But now, I think it's good to understand where do all these parameters go? You train a net with a million parameters, but it acts like a net with fewer parameters – where do these parameters go? I think these may have to do with the other mysteries as well. Complicated data needs more parameters maybe, and that's made formal by generalization theory. The basic theorem of generalization theory say something like the following:

$$\text{test loss} - \text{training loss} \leq \sqrt{\frac{N}{m}}$$

where N is the "effective capacity", like VC-dimension etc and m is the number of samples. Now the simplest bound on N is just the number of model parameters, which is vacuous if $N > m$. Classic results replace N by VC-dimension, Rademacher complexity, etc. It's known these measures are bad for deep nets.

There's an old notion that flat minima generalize well. Recently this was tested empirically by Keskar et. al. (2016). Intuitively, why should flat minima have a lower number of parameters? It's a description length idea. You don't have to specify as many bits of accuracy, so it has smaller description length and a fewer number of parameters. So this is one of the many mnay qualitative theories of deep learning. But can you make this quantitative? Do the calculation that's implied by this description and give a bound on the effective number of parameters? That is a tough problem. Let me tell you something simpler: Linear classifiers generalize well, if the linear classifier separates the data and has a margin. That's a bit like a flat minimum, because the minimum is this hyperplane separating the data, and there's some fat slab allowed in specifying the hyperplane. That's like a flat minimum. You might imagine you can represent this hypothesis with a lot fewer numbers, and indeed you can express this with $\log N/\gamma^2$ numbers. You can do this calculation many ways. So you only need that many samples.

The second way to make things quantitative was using PAC-Bayes bounds, from Langford and Caruana. What's the analysis notion of margin for a deep net? They suggest the following notion: Suppose θ^* is the deep net you have, and you can imagine adding Gaussian noise to all the parameters. Do the trained net and the noised net have similar training noise? Then you think there is a flat minima, a sea of other deep nets around θ^* which are almost as good. You can do this and get some estimate of effective capacity using the variance of the noise. This is connected to the PAC-Bayes theory of McAllester 1999. But it's very hard to get non-vacuous bounds.

We used these methods to get some bounds, and the true number of parameters is much less than the bounds. Dziugaite-Roy have non-vacuous bounds for MNIST but don't get any "complexity measure". We get estimates that are of the order of parameters, but it's still not 50000 – but at least we get something slightly non-trivial. How is this new result proven?

Question/Answer: Can we exploit a clean mathematical model of what an image is? One should probably start with a toy model of images, some wavelet kind of thing.

We introduce a new kind of noise stability, which gets attenuated as you pass through higher layers. Our notion of margin is "noise stability". You add a lot of noise, nevertheless, the higher layers reject the noise. The noise starts off being as high as the noise of the layer, but it gets attenuated as you go deeper. We think this property had not been noticed before. This may remind you of von Neumann (probabilistic logic and reliable gates from unreliable components), as well as Shannon – we have human brains which seem reliable, constructed from very unreliable components. They were able to solve this via information theory and redundancy. So you might think that the networks are highly redundant. We still need to show that you can compress a neural network mathematically.

Suppose we have a single layer with no non-linearity. We have a layer x, and map it to Mx. Now add noise, $x + \eta \to M(x + \eta)$. Noise stability means that $|Mx|/|x| >> |M\eta|/|\eta|$ for one layer. The largest it can be is $\sigma_{\max}(M)$. And noise just spreads itself across all directions uniformly: ℓ_2 norm of singular values scaled by $\sqrt{n}$. So the left side is much larger than the right hand side. So it's saying something linear algebraic about the layer. You can verify the spectrum of each layer. We call this ratio the layer cushion, our theory assumes that this ratio is good. The compression is going to use this ratio. The idea is that x will align with these singular values.

Noise stability implies that deep nets are highly redundant, and we are going to compress it. Now, compressing neural nets is a very active area in applied research because they want to compress nets to put them into a cell-phone. So people have all kinds of methods empirically to compress neural nets. So this is just one with some provable guarantees.

So: take a layer and randomized compress it. We can show the errors introduced in the error are Gaussian like, so these errors attenuate as they go through the layers, so you can keep compressing the other layers and the errors don't get big. The gaussianity doesn't seem important: We believe the errors are gaussian so we pick a particular compression scheme, you could do it with others depending on what you think the noise looked like.

We have something called an interlayer cushion: You take $f(x) = \nabla_x f \cdot x$. If you evaluate it at points which are not x, it becomes something else than just the Jacobian. In a small perturbation, it looks similar to the Jacobian. The third parameter is interlayer smoothness – how well does the Jacobian approximate the function in the neighborhood. So then we have a generalization bound in terms of $\approx$ (depth $*$ activation contraction/layer cushion $*$ interlayer cushion)^2. These are all computable measures on the training data.

To compress: Generate k random sign matrices $M_1, \cdots, M_k$ (important: pick before seeing data). Then multiply.

The single-layer cushion is the real driver of this whole theory. Now do these concepts correlate to generalization? If you have corrupted data, does this generalize well? You see the bound improves if you keep training past the stage that the test error has gone down.

Thus far, we were only talking about fully-connected nets. You can fully extend to convolutional nets. The difficulty in convolutional nets is that they already are re-using convolutional parameters – in some sense, the net is already compressed. So first, you might try to compress each copy of filter independently – this blows up the effective number of parameters. You could also try compress each filter once and re-use it over the image. It's a bad idea in theory, because now you produce errors which are correlated across the image. In theory, you'd like these errors to be uncorrelated. So our idea was to use p-wise independent compression. It's just an idea from pseudorandomness.

1.5 Conclusion

So these are only first-cut analyses. These are just the beginning of a theory of deep learning. Much remains to be done. There are other things that would be nice to have theory for: Deep RL, deep models for language understanding, generative nets, deep generative nets. There are lots of other aspects of deep learning I didn't talk about today. There will be a special year at IAS in $2019 - 2020$. If you're interested in visiting, please contact us. We have a blog called offconvex.org. I have a grad seminar from last fall, and other resources.

1.6 Questions

Just a disclaimer: I don't think compression idea fully explains 50000 samples and 6 million parameters. Do you think other compression algorithms for deep nets are further compressible using this approach – Maybe, it's a good question.

2 On the Optimization Landscape of Neural Networks (Joan Bruna)

We are going to think about the loss surface in terms of its level sets. One thing we are going to be studying is very simple topological properties of these sublevel sets, for instance, whether the level sets are connected or not. If there's only one connected component for each energy level, there cannot be any poor local minima. More generally, we are interested in certifying existence of descent paths.

A valley is a connected component of a sublevel set Ω_u. A spurious valley is a connected component if it does not contain a global minima. If a loss has no spurious valley, then on can continuously move from any point in parameter space to a global minima without increasing the loss.

Our result says that in the "deep linear network case" not only does our model have no poor local minima, then there is only one value for every set of connected components. We have

$$E(W_1, W_2) = \mathbb{E}_{(X,Y)\sim P}\left[\|W_2W_1X - Y\|^2 + \lambda(\|W_1\|^2 + \|W_2\|^2)\right]$$

satisifes $N_u = 1$ if $n_1 > \min(n, m)$, where N_u is the number of minima.

The idea is we do an induction on the number of layers, and re-parametrize the model so that we move to "canonical parameters". You can think of this as a single layer model. We lift the parameter space to $\tilde{W} = W_1W_2$. This problem is convex, and there exists a linear path $\tilde{\gamma}(t)$ that connects Θ^A, Θ^B. Write the path in terms of original coordinates by factorizing $\tilde{\gamma}(t)$. A simple fact is if $M_0, M_1 \in \mathbb{R}^{n\times n'}$ with $n' > n$, then there exists a path which goes to a global minimum.

How much extra redundancy are we paying to achieve that $N_u = 1$ instead of simply no poor local minima? We can look at the multilinear case. You can construct an equivalence class in parameter space, by writing $\tilde{W}_k = U_kW_kU_k^{-1}$ for the case where we look at $W_1 \cdots W_K$. This doesn't change the value at all. Now you have to work on the subspaces of the Grassmanian space, and work on the quotient space.

This is all linear case. If we look at quadratic nonlinearities, we can linearize it easily by looking at outer products: $\rho(Wx) = A_WX$, where $X = xx^T, A_W = (W_kW_k^T)_{k\leq M}$. Level sets are connected with sufficient overparametrization: If $M_k \geq 3N^{2^k}$, for all $k \leq K$, then the landscape of K-layer quadratic network is simple: $N_u = 1$ for all u.

Good behavior is recovered with nonlinear ReLU networks, provided they are sufficiently overparametrized. We can't prove there are no spurious valleys, but we can "almost" connect level sets, with some jump in energy, but this goes to 0 slowly with exponent 1/dimensionality. This is a consequence based on local linearization of ReLU; not being able to write it as a kernel.

So really what we are trying to do is write the functional space of the neural nets as a kernel reproducing space:

$$\phi(X; \Theta) = \langle \Psi(X), A(\Theta)\rangle$$

where $\beta = A(\Theta)$ are the canonical parameters and Θ is the neural net parameters, and $\Phi(x; \Theta) = W_k\rho(W_{k-1}\cdots\rho(W_1X))$.

Theorem 2.1. *If dim*$(A(w), w \in \mathbb{R}^n) = q < \infty$*, then* $L(U, W) = \mathbb{E}\left[\|U\rho(WX) - Y\|^2\right]$ *has no spurious valley if* $M \geq q$.

This includes ERM since RKHS is only queries on a finite number of data points.

We can also think about the problem in functional space generated by the model. We have $F_\Phi = \{\phi : \mathbb{R}^n \to \mathbb{R}^m; \phi(x) = \Phi(x; \Theta) \text{ for some } \Theta\}$. Sufficient conditions: F_Θ is convex and Θ is sufficiently large so we can move freely within. A necessary condition is that F_Φ is ball-connected, where if you intersect the space with balls, there's only one connected component. This is a bit more general than convex. What if it's not sufficiently overparameterized. Here we can hypothesize that energy barriers of spin-glasses are important. You look at a Gaussian polynomial over the sphere. It has a bunch of critical points all over the place, organized by at what energy they emerge. Above the energy barrier, the probability you find a poor local minima is small. With high probability you always end up with this energy barrier. Can we have the same kind of phenomenon in our setting? We are working on proving that the loss surface has no poor local minima above the energy barrier.

We also want to study what happens in practice. What kind of conditioning is necessary for gradient descent? Some kinds of level sets might be better to optimize over. The idea is that more movement is harder (more turning etc.). We estimate geodesics of level sets, and look at their length. We are going to try to make this path as short as possible. We need to find the shortest possible path in the level set. Maybe we hope for a linear path. If it's in the set, we're done. Otherwise we need to perturb the path, until it falls into the level set. We penalize how far we move. Then you just keep iterate the same thing in a dynamic programming approach. This gives us a measure that's intrinsic. For real problems with real networks, we measure this. We see that MNIST is essentially convex with a CNN. If you do this on more complicated data, you see similar things.

We can show that there is no spurious valleys, no disconnectedness. This suggests there's a lot of room to explain what happens in practice. Is there a sweet spot between overparameterization and overfitting? Check out Topology and Geometry of Deep Rectified Network Optimization Landscapes, and Neural Networks with Finite Instrinsic Dimension Have no Spurious Valleys.

3 A theory of deep learning dynamics: Insights from the linear case (Andrew M. Saxe)

Is deep learning hard to train? We'd like to understand the time course of training and what does depth contribute to the learning dynamics? Are there other important features controlling training speed? How do deep networks generalize dynamically? We'll look at deep linear networks.

Particularly for brain sciences, it's important to have a simple model.

This talk characterizes the training and generalization dynamics of deep networks, looking at optimal stopping times and things like this. See https://arxiv.org/pdf/1710.03667.pdf

In high-dimensional regime, small initial weights are critical for good generalization (frozen subspace not modified, these weights never change! so ideally the area that's not modified should have small effect). This is a key difference from the deep network area (frozen subspaces are frozen for shallow nets, you can move a bit for deep networks). Overtraining is catastrophic when dataset size is matched to the number of parameters in the model.

The more accurate labels, the longer you're justified in training.

Deep networks likely operate in the undetermined accurate label regime where overtraining is not a problem. Correct answers are deterministically labeled.

This might be another evidence for NLP not working as well for certain kinds of tasks, where things are noisy labels by nature almost.

The approach I took was minimal models – how much can we take away things. Look at linear models for instance.

4 Towards understanding the invertibility of convolutional neural networks (Anna Gilbert)

We started with the observation that you can nearly perfectly reconstruct images from a bunch of features, assuming that you retain the "where" values of the nonzeros where you did max-pooling. So this means that CNNs, assuming you keep certain information, are invertible. We provide theoretical analysis about the invertibility of CNNs. CNNs can be analyzable with a form of compressed sensing, and you're essentially doing sparse signal recovery in a certain fashion. You can thus get a reconstruction bound on how well these networks do.

The theoretical result is that transposed convolution satisfies a model-RIP property.

The convolution part of a net is a set of filter matrices times a vector, ignore the pooling for now. ReLU says set negative coefficients to zero, and retain the others. Suppose you had $+W$ and $-W$, and you multiplied that vector times x. If you applied ReLU, you'd have the positive part, and - the negative part: $f(x) = \max(0, f(x)) - \max(0, -f(x))$. This is purely splitting the vector into positive and negative components. If you could check to see if you had positive and negative filters in the CNN, you could get rid of the whole ReLU business. Is that assumption a reasonable thing to do? It turns out that this is mostly the case for all layers. So we don't need to worry about ReLU. Now what about convolution and pooling? We're going to keep pooling, as long as we retain the positions of the max coefficient over each pool. The decoder network says "unpool": put in the location of the max pool and fill the rest with zeros. Then finally deconvolve (multiply by transpose of convolution matrix).

So how does compressed sensing come into it? Basically, the deconvolution process satisfies a certain kind of RIP condition.

In order to check this, we need to check that outputs of CNNs are sparse, and we also need to check that the deconvolution procedure is multiplicative. This is the math part. Finally, is a Gaussian random filter assumption reasonable? We do this part with data analysis. They look at what trained filters look like to empirically argue that it looks like RIP. Model-RIP is a sparsity requirement over chunks: this is exactly like max pooling.

So, we can view pooling as producing a sparse vector. ReLU contributes to the sparsity. Convolution is just a linear operator, for multiple channels just concatenate. Assuming filters are Gaussian doesn't quite finish the argument, because you have short filters – things concentrate worse. Having more filters lets you concentrate.

Reconstruction error bound for iterative hard thresholding (model-RIP version, per iteration step bound):

$$\|\hat{x} - x\|_2 \leq \frac{5\delta_{2k}}{1-\delta_k}\sqrt{\frac{1+\delta_{2k}}{1-\delta_{2k}}}\|x\|_2$$

5 On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization (Nadav Cohen)

Why depth? What does depth give us? The conventional wisdom is that depth boosts expressive power. Depth allows us to represent much more compactly functions which we would otherwise pay a more expensive price. There is a price though: optimization becomes more difficult. We argue in this work that depth can actually accelerate optimization, even though things are non-convex.

The common approach to analyze optimization in deep learning is to characterize the landscapes – in particular, the critical points (local minima/saddles). This kind of approach by definition prefers convex objectives. Nothing is simpler with an objective with a single critical point that's a global optima, if you view the problem through critical points. But if you want to care about depth, you have to consider the dynamics of the optimization algorithm, and how it removes in parameter space. An immediate hurdle arises: expressiveness can interfere with our study. If we naively compare deep network to a shallow one, it is not obvious whether this is a result of some inherent acceleration phenomena, or whether the deep network can fit the training data better. To address this hurdle, we focus on linear networks – when you add layers, you don't change anything in terms of expressiveness. If deeper linear nets converge faster than shallow ones, this has to be the result of favorable properties of depth for optimization. You essentially replace a matrix parameter with a product of matrices – it's an overparametrization.

Consider ℓ_p regression: $L(w) = \sum_{(x,y)\in S} \frac{1}{p}(x^T w - y)^p$. See the discussion by Sanjeev from before (replace $w = w_1 * \omega_2$) – this is completely redundant. We're going to see that this has a vast effect on the dynamics of gradient descent. The gradients look like for w_1, and similarly for ω_2, the following:

$$\nabla_{w_1} = \sum_{(x,y)\in S} (x^T w_1 \omega_2 - y)^{p-1} x \omega_w$$

Gradient descent requires two updates now. Assuming learning rate is small, we can write down the dynamics of the end to end model:

$$w^{t+1} = w^t - \rho^t \nabla_{w^t} - \sum_{\tau=1}^{t-1} \mu(t,\tau)\nabla_w(\tau)$$

for suitable ρ, τ varying over time.

The mere introduction of a simple scalar turned gradient descent into a scheme that involves an adaptive learning rate and some kind of a momentum. This emphasizes the vast effect that overparameterizing can have. Now let's be more formal.

We will analyze deep linear networks: $x \to W_N W_{N-1} \cdots W_1 x$. Let $W_e = W_N \cdots W_1$. Recall this is still just a linear model. Given some loss over the linear model, we have overparametrized loss

$$L^N(W_1, \cdots, W_N) := L(W_e)$$

How does the end-to-end matrix behave when we use gradient descent over $W_1, \cdots, W_N$? We have

$$W_j^{t+1} \leftarrow W_j^t - \eta \frac{\partial L^N}{\partial W_j}(W_1^t, \cdots, W_N^t)$$

Theorem 5.1. *End-to-end update rule.*
Assume we have a small learning rate η. Then,

$$W_e^{t+1} \leftarrow W_e^t - \eta \sum_{j=1}^{N} [W_e^t (W_e^t)^T]^{(j-1)/N} \frac{dL}{dW}(W_e^t)[((W_e^t)^T W_e^t)^{(N-j)/N}]$$

We see that the gradient undergoes some transformation: It's multiplied by different matrices from the left and right, and there's a summation. Both are PSD matrices, and they are brought to different powers. This is how the end-to-end matrix behaves when you apply gradient descent over a linear network. For now, we see that overparametrization with a deep linear network gives rise to an update rule for which we have a closed form expression. This update rule doesn't depend in any way on intermediate dimensions. It doesn't matter what the dimensions were for the series of matrices $W_1, \cdots, W_N$. So here, the width of the network is insignificant, not the depth. There is still a momentum term here: The memory here (in terms of momentum) is embedded into the location in parameter space.

How do we prove this? The assumption that $\eta << 1$ implies we analyze discrete updates through differential equations. The assumption that $W_j^0 \approx 0$ is for the assumption that $(W_{j+1}^0)^T W_{j+1}^0 \approx W_j^0 (W_j^0)^T$. If this condition holds at initialization, then it will continue to hold throughout the entire optimization. We show it continues to align for all t: Then the left singular spaces of W_j^t coincide with the right ones W_{j+1}^t, which allows us to derive the closed form update rule.

So to what extent do these hold in practice? What we do is evaluate this empirically. You can see here 2-layer and 3-layer networks. Each network is compared to its equivalent analytically derived update rule for a deep linear network. The analytic update rule indeed complies with deep network optimization.

What does the end-to-end update rule mean? If you arrange the weight matrix as a vector, and re-write the update rule, it assumes the following form:

$$vec(W_e^{t+1}) \leftarrow vec(W_e^t) - \eta \cdot P_{W_e^t} vec(\frac{dL}{DW}(W_e^t))$$

where $P_{W_e^t}$ is a preconditioning PSD matrix. This term is the only difference from gradient descent. We are optimizing a matrix through this update rule, and arranging it as a vector: This is normal gradient descent with a preconditioning matrix, varying throughout the optimization, depending on where we are in parameter (matrix) space. The singular vectors give rise to eigendirections in the preconditioning. If their presence in the matrix is strong (high singular values), then the direction will be accompanied by a large eigenvalue. So it stretches directions according to location in parameter space. Since we're close to 0 in the beginning, that means that the overparameterization induces a certain preconditioning in movements we've already taken. This is the implicit effect of depth.

Let's consider a special case and be more concrete. Consider the single output case with a linear model: Output a scalar from a vector. In this case, the update rule looks like

$$W_e^{t+1} \leftarrow W_e^t - \eta \|W_e^t\|_2^{2-2/N} \left(\frac{dL}{dW}(W_e^t) + (N-1) Proj_{W_e^t}(\frac{dL}{dW}(W_e^t)) \right)$$

The $\|W_e\|_2^{2-2/N}$ term is an adaptive learning rate, increases as you step away from initialization. The term $(N-1)Proj_{W_e^t}(\frac{dL}{dW}(W_e^t))$ is a kind of momentum: It favors the direction taken so far (it is like the pull back in Nesterove momentum). So this update rule doesn't have any memory explicitly encoded here. Now there's a question: Can this optimization scheme be realized as gradient descent over some other objective, via for instance with some kind of regularization? Regularization is useful in optimization to speed things up, it's not just related to generalization.

Theorem 5.2. *Assuming $\frac{dL}{dW}(0) \neq 0$, there is no function of W whose gradient is given by this form. Overparametrizing by depth gives you a way that you can't get by regularizing.*

Proof. By fundamental theorem for line integrals: $\int_\gamma \nabla g = 0$ for any closed curve γ and differentiable g. We can compute bounds on the line integral over this curve, and show that this lower bound is positive when radius is small enough. That means that the vector field contradicts fundamental theorem of line integrals, and thus cannot be a gradient of anything. □

But so far, we don't know whether this is a good thing or not! Does it help? It turns out the answer is complicated. Sometimes it helps, sometimes it doesn't.

We can look at classic ℓ_p regression in 2 dimensions. If one of the coordinates of the correct answer is ill-conditioned in the sense that one coordinate is much smaller than the other (for the correct answer), the η you pick depends on $2/\max(y_1, y_2)$ – the larger one dominates the learning rate, and makes it really small, causing everything to slow down for all coordinates. with overparametrization, the learning rate gets multiplied by $(w_1 + w_2)^{2-2/N}$, and as one coordinate converges slowly, the other coordinate doesn't experience the slow-down as well.

We did experiments with depth 1-hidden layer linear nets (basically no computational overhead). We find that overparametrizing with a single scalar significantly accelerated ℓ_p regression for $p > 2$! It also improves over AdaGrad and AdaDelta. We artificially nonconvexified, it now speeds things up! Gradient descent on the nonconvex objective improves over dedicated convexification. The more ill-conditioned the problem is, the greater the advantage will be. Adam was faster than overparametrized gradient descent, but we saw that overparametrized Adam is better than regular Adam empirically! So this might not be limited to just gradient descent.

Note that learning rate is very important: We do a grid search about 20 values of learning rates; we present the fastest best results for each problem.

There's now a natural question: Why not go deeper and deeper if it accelerates? The answer is the vanishing gradient problem. The term $\|W_e^t\|_2^{2-2/N}$ multiplies, and if N is large, multiplying a lot of small things is very small. Thus, the learning rate is effectively brought down to zero, and you're stuck when you start off. The deeper your network is going to be, the more your problem will be. People usually initialize weights to be larger: specifically, identity initialization is good – this results in linear residual networks (Tengyu Ma and Moritz Hardt). This allows acceleration for up to 8 layers, we did not go deeper.

Also note that these things are all with regular gradient descent, not stochastic.

We also took the off the shelf Tensorflow ConvNet tutorial to see the effect of overparametrization. To each dense layer, we added an excess matrix. This modification does not change expressiveness in any way. We just plugged in these extra two matrices. The improvement that we got is very significant. With only adding 15% more parameters, overparametrization accelerated non-linear nets by orders of magnitude. Other settings also showed acceleration.

The objective that the overparametrization sees is not exactly cross-entropy, and it changes over time. So we do not have an analysis for this yet. We would like to adapt the analysis to specific non-linear models (for instance tensors), and are thinking about it now.

So, depth radically changes the objective landscape, and makes things highly non-convex. This is somehow practically possible to get by with, but we don't understand it yet. Conventional wisdom is that it makes optimization more difficult. But the objective landscape approach reinforces this conventional wisdom, by definition (more critical points = worse). Here we show that overparametrization results in a combination of adaptive learning rate and inertia, cannot be attained through regularization, and significantly can lead to acceleration. It doesn't always, but in many cases, it does, and the computational overhead associated with the acceleration can be negligible.

Approaching deep learning optimization through the lens of nonconvex analysis biases us towards worst case scenarios that seem to not happen. It appears that for linear networks, there are saddle points. Yet, when you analyze the behavior of gradient descent, it seems to avoid them. The *dynamics of the optimization algorithm matter*. You can't understand optimization in deep learning without taking this and the specific models into account. I believe this is an approach which might help us to understand.

Note that this is a linear framework (overparametrized), and we didn't touch on generalization at all – just optimization. Also see the blog post on this topic.

5.1 Questions

How to get the optimal number of layers, if you don't initialize with identity? In practice, the transition from 3 to 4 layers makes it much slower.

Can one also rule out Newton steps or other Hessian based optimization methods as possible forms for the overparametrization? For adaptive gradient steps, the question doesn't make sense. I have to think about it for the case of Newton, it's an interesting question.

Why are ℓ_p for $p > 2$ losses amenable to acceleration why $p = 2$ isn't? I didn't catch the theoretical reason for this: The answer is that there's a semi-theoretical reason: If p is larger, the order of the norm by which you multiply the derivative by is larger, so it's more possible to use more layers which helps out (you can start out with smaller initialization and still have the gradient not vanish, I think – need to check this in the paper/blog post).