Algorithms-ML Seminar

Why Deep Nets are Reversible: A simple theory

November 19, 2015

Lecturers: Sanjeev Arora, Tengyu Ma, Yingyu Liang   Scribe: Kiran Vodrahalli

Contents

1 Overview

This work introduces a reversible generative model for a full neural network with ReLu gates. Reversibility is derived by showing similarity to the behavior of Gaussian random matrices. In addition to introducing a good theoretical model which allows you to further analyze the behavior of neural nets, this gives you extra training samples by reversing the net at each step. This training method empirically outperforms dropout.

2 Introduction

Neural nets are basically a circut where you have an input x, and transform each layer by doing $r(Ax + b)$. There is a final output layer z (which is not a bit, rather it is an output vector of real numbers), where you get a loss and a gradient. You update backwards by doing chain rule. Here r is a rectified linear function (ReLu). The problem is that these tend to require a lot of samples. People have been trying out generative models to model this phenomenon. The goal is that you get a probabilistic model which is a distribution you can sample from: $\mathbf{P}\{x|z\}$, or even nicer: $\mathbf{P}\{x, z\}$. Google and everyone else are basically trying to come up with generative models for images (you see this in the news).

The point is that in Bayesian modeling, and in machine learning in general, people do not believe in NP-completeness. Once you have a closed form and you can generate things, you can do all kinds of inference. This is why this is seemingly useful. If you can learn this model $\mathbf{P}\{x|z\}$, then your output is just $\mathbf{P}\{z|x\}$. This would basically be a substitute for the neural net. Empirically you can generate $\mathbf{P}\{z|x\}$, even though it is $\#P$-hard.

There are two points: (a) we want a model, (b) the feedforward deep net should compute $z|x$ via MLE/MAP in the new model.

There are also non-generative approaches, which is basically reconstruction error. If you are not a Bayesian, you can just say you are trying to come up with a method that reconstructs z given x. This is more of a philosophical question, where you either interpret these values as probabilities or quantities to optimize. You can basically write down the same expression as a reconstruction error, and there is no difference (for instance, KL-divergence). An example is RBM: You have $\mathbf{P}\{x, h\} \sim \exp(-x^T Ah + hh^T)$ for one layer of a Restricted Boltzman machine, with input x and output h. This is an energy based model, which is nonconvex, but people still try to fit it. In this form, it is easy to sample from since $\mathbf{P}\{h|x\}$ is a sigmoid. They use Bayesian optimization to sample from the joint probability. In feedforward and feedback, it is basically a probabilistic alternative to the neural net in the setting where gates are sigmoids, not ReLu. The trouble with these methods is that they could not define an end-to-end probability distribution. There is no way to make this consistent. The reason is that the marginal distribution in one direction for x will in general be different from the reverse distribution. They basically took a one-layer idea and extend it to multiple layers, but that doesn't work. So people don't have consistent multiple models.

So what are the crazy pictures Google generates and displays in the New York Times? They are training an external model on images. These days people think of neural nets as a non-convex optimization **technique**. They learn generative models using neural nets (called adversarial nets). Using these, you can get the Google Dreams and so on. You can take an existing picture, map it, find the hidden variables, and change the variables, so now sheep are in the sky and so on.

We want to show that our feedforward deep net somehow computes $z|x$ (via MLE, MAP, etc.) in this model.

Claim 2.1. *Random-like net hypothesis.*
A layer behaves like a random matrix.

Definition 2.2. Universality.
Bulk properties are similar to $A \in GUE$, Gaussian Unitary Ensemble. For instance Ax and x are uncorrelated for almost all x, due to eigenvalues. There are other properties reasonable for random matrices which can be realized by neural nets.

If it is like a Gaussian, there is a trivial sort of model. The forward direction is Ax. If you just reverse the network (i.e. $A^T(Ax) \approx Ix$ where A is a random matrix). So it approximately reverses for trivial reasons if it is randomized. Now the idea is that it's a nonlinear transformation, well it turns out that this also reverses. It is not one-to-one, one-to-many. You can't be one-to-one in both directions. Using A^T, you can define an inverse of sorts using ReLus. It turns out these correct for noise. Suppose you have this now. What good is this?

If you are trying to train a network which has this random-like property, you would expect if you feedforward x, you get some z. Then when you reverse, you get $z \to \tilde{x}$. By the property of the model, $x, \tilde{x}$ map to the same z. Therefore according to the net they both have the same label. Now you can take $\tilde{x}$ and add it to your training set: $\tilde{x}$ has the same label as this. At each epoch, you generate some new synthetic example. Now your set will be double the size. Does this work? Yes, it does, and provides some visible improvement over Dropout empirically, which is the standard way to train neural nets. So the theory now explains Dropout, ReLu, and also this generative part which boosts Dropout by a percent or so.

3 Technical Details of the Proof

Let us talk about one layer for now, but keep in mind that the goal is to make this extendible to multiple layers. Suppose we have a hidden variable $h \in \mathbb{R}^n$. We are trying to generate an m-dimensional vector $x \in \mathbb{R}^m$, observable. We have $W \in \mathbb{R}^{m \times n}$ and we would like $\mathbf{P}\{x|h\}$. The linear model is simply $x = Wh$ - a completely deterministic model. We also take $W \sim \mathcal{N}(0,1)$, which means that each entry of W is Gaussian and independent.

Theorem 3.1. *$\|h - \frac{1}{m}W^Tx\| \leq \epsilon\|h\|$, where $W^Tx = W^TWh$ and we have $\|h - \frac{1}{m}W^Tx\| = (I - \frac{1}{m}W^TW)h$. The matrix here has spectral norm less than $\sqrt{n/m}$. By applying matrix concentration, we get that this is all $\leq \sqrt{n/m}\|h\|$ for any h.*

Now we give our generative model. $x = r(\alpha W h)\dot{\eta}_{drop}$ where dropout is multiplying termwise and η_{drop} is 0 (zeros it out) with probability $1-p$, and leaves the entry alone with probability p.

Theorem 3.2. *$\exists b$, a constant scalar, such that for most of W, and for most of k-sparse h, if you look at $\|h - r(W^T x + b)\| \leq \epsilon \|h\|$, where $\epsilon = \sqrt{\frac{k}{pn}}$. Basically pn is the expected sparsity of x. This is basically saying you need the layer below to be denser than the layer above.*

Lemma 3.3. *$\|h - W^T x\|_\infty \leq \epsilon \cdot \frac{\|h\|}{\sqrt{k}}$. This is the average magnitude of the entopy of h. $W^T x$ is already close to h in the infinity norm. If you set $b = -\epsilon\frac{\|h\|}{k}$, and set $i \in [n]$ such that $h_i = 0$, then you can check that $r(W^T x + b)_i = 0$. You get exact recovery in the situations where this is non-zero (you are in the support of h), this is the denoising property of ReLu. In other words, $|r(W^T x + b)_i - (W^T x)_i| \leq b$ for $i \in supp(h)$. Essentially there is a gap between the scrambling of the large coordinates and small coordinates. This is similar to compressed sensing, except this is nonlinear.*

You do in fact recover the layer above if you do this on any neural nets. You do actually recover the layer above. ReLu removes information about the negative part, but you can still recover.

Proof. We will do this coordinate by coordinate since we are in the ∞-norm. $(W^T x)_i = \sum_j W_{ji} x_j = \sum_j W_{ji} r(\alpha \sum_l W_{jl} h_l) = \alpha \sum_j W_{ji} \left(r(W_{ji} h_i + \sum_{l \neq i} W_{jl} h_l \right)$. If you have no ReLu, you would get $\alpha \sum_j W_{ji}^2 h_i + W_{ji} \sum_{l \neq i} W_{jl} h_l$. The second term is basically noise, and the first term is the signal. Dropout is just a subset of the sum, so here we assume no dropout. Each term in the noise term has standard deviation $\sqrt{k}$. Thus in the no-ReLu case, this would equal $\alpha(n/2)h_i + \sqrt{k}\sqrt{n})$. This is why you can recover h_i. This is the universality condition. Universality is about one quadratic expression, eigenvalues. The one we have here, in a real matrix behaves as it would in a random matrix. The ReLu is to keep multiple layers sparse. Also you need the nonlinearity, otherwise you would just have all linear functions. For one layer it is not needed. Now suppose you have ReLu. Intuitively it is not too different. You can see $r(W_{ji} h_i + \sum_{l \neq i} W_{jl} h_l)$. You can imagine that the sign of this whole thing is dominated by the latter term, call it η_j. Assume this is approximately $\mathbf{1}_{\eta_j > 0} \cdot (W_{ji} h_i + \eta_j)$. The only case where it is not equal is when $\eta_j > 0$ and $W_{ji} h_i + \eta_j$ is negative, which happens with small probability. Since η_j has standard deviation $\sqrt{k}$, and the other one has standard deviation 1. If you take expectations you get a good approximation (the bad case happens with less than $\frac{1}{\sqrt{k}}$ probability). Again you can write $\mathbf{1}_{\eta_j > 0} \cdot (W_{ji}^2 h_i + W_{ji} \eta_j)$. If you want to do a better approximation, you need to do a Taylor expansion. You need Lipschitz to give this property. □

This lemma only gives the $\| \cdot \|_\infty$ bound, to get to the $\| \cdot \|_2$ bound, you need some kind of flatness. You need a thresholding scheme.

Now that we have a model that's reversible, under this setting you could try to prove that backprop converges in the nonlinear case. That is what we are currently working on for one layer.

4 Experiments

We use our generative models to generate images. After training, you move towards a location such that your generative model is getting closer and closer to the true distribution. You just look at the layer before the final logistic softmax. This even works with random nets. It is basically a noisy regularizer. You take the real image, use the net to compute the hidden variable (top layer), and then you use the net again to generate the image (reverse the net). The hidden representation is sparse (this is compression basically). These are not convolutional nets, so the predictive accuracy doesn't mean much here. The theory can probably be extended to convolutional nets, but we have not done this yet.

The testing error of the network with the new method for training (denoted SHADOW, i.e. using the $\tilde{x}$ samples as well). They get faster convergence and better test error than just plain dropout.