Mathematical Aspects of Deep Learning

Mathematics of Deep Learning: Lecture 8 – Hierarchical Generative Models for Deep Learning

Transcribed by Frederic Koehler (edited by Asad Lodhia, Elchanan Mossel and Matthew Brennan)

Today we will study a model interpolating between tree reconstruction (introduced last time) and deep learning (topic of this class).

Recall the model last time: we have a broadcasting process on a tree T with root r and ℓ levels, where k samples, each of the form $\{\sigma'(v)\}_{v\in\text{Nodes}(T)}$ are generated by the following process:

1. Sample $\sigma'(r)$ for the root r uniformly from $[q] = \{1, \dots, q\}$.
2. Given $\sigma'(w)$ and v is a child of w, then with probability θ set $\sigma'(v) = \sigma'(w)$ and with probability $1-\theta$ draw $\sigma'(v)$ from the uniform distribution on $[q]$.

We let $\sigma(v)$ be the k-tuple of the k i.i.d. samples of $\sigma'(v)$. Last time we discussed how you can reconstruct the structure of T given $\{\sigma(v)\}_{v\in\text{Leaves}(T)}$.

A Model Between Tree Reconstruction and Deep Learning

We will now introduce two models that build on one another and interpolate between tree reconstruction and deep learning.

1. First Step: Semi-supervised Learning Model

The first step in our interpolation is to convert the problem into one of semi-supervised learning. We attach to each vertex v in the graph a set of labels $L(v)$, with the constraint that is v is above w then $L(v) \subset L(w)$, i.e. we only accumulate labels as we go down the tree. Furthermore we require that if $a \in L(v_1) \cap L(v_2)$ then the lowest common ancestor v of v_1 and v_2 satisfies $a \in L(v)$.

The motivation for this comes from evolution: we should think that the labels consist of tags like "Dog" and "German Shepherd" (in which case the $\sigma(v)$ at the leaves model the genetic data of each species), and the rules give that every descendant of a dog is a dog and that every two dogs most common ancestor was also a dog.

Now that we have labels we can consider the following semi-supervised learning task: the data is $\{\sigma(v)\}_{v \in Level(\ell)}$ and $\{(\sigma(v), L(v)) : v \in S\}$ for some subset S of the leaves ($Level(\ell)$). Our task is to infer $L(v)$ for all $v \in Level(\ell)$. Roughly speaking this models the problem that we have the genetic code for many animals, some of which are labeled by tags such as "Dog" and "Cat" etc. and now we want to infer for all of the unlabelled animals whether they are a dog or cat as well.

2. Second Step: Adding Nonlinearity

Now we move on to the second step: we want to introduce nonlinear functions of multiple variables (like the sigmoid composed with linear map in the neurons of a neural network). We modify the way $\sigma(v)$ is generated given its parent $\sigma(w)$:

1. For $i = 1, \ldots, k$, let $\tilde{\sigma}(v)_i$ be $\sigma(w)_i$ with probability θ and uniform at random from $[q]$ with probability $1-\theta$.
2. Define

$$(\sigma(v)_{2i-1}, \sigma(v)_{2i}) := \Pi_{(w,v)} \left(\tilde{\sigma}(v)_{\Sigma_{|v|}(2i-1)}, \tilde{\sigma}(v)_{\Sigma_{|v|}(2i)}\right)$$

where $\Pi_{(w,v)} : [q]^2 \to [q]^2$ is a function depending only on w and v, and $\Sigma_{|v|}$ is a permutation on $[k]$ depending only on the level of v. (Note in particular that neither of these depend on i).

Thus the variable $\sigma_i(v)$ is a function of noisy copies of two different entries of $\sigma(w)$. The ability to choose Σ gives us freedom in which of these two entries they are, for example σ_i could depend on one of positions i and $i + 2^r$ in its parent and this roughly models the way convolutional neural networks mix across different length scales in each layer.

Also note that applying just one $\Pi \in S_{q^2}$ is already interesting; i.e. how can we recover the root from the leaves in a known tree? For example if Π is drawn uniformly at random, and only at the last layer, and $k = 1$ then by symmetry $\sigma(r)$ is independent from the values of σ at the leaves which is markedly different from the situation without Π.

Roughly the analogy to a standard neural network is as follows: Σ represents the connections of neurons, Π is the nonlinear activation composed with a linear function and the noise is a "nuisance variable" that makes objects different (instead of them just being copies of each other). For example when $\Sigma = Id, \Pi = Id$ and there is no noise, the network is just copying. We can also visualize a sort of "deep net" of a leaf where we look at the interchanges of the variables (given by Σ) between σ on the path from the root to the leaf.

Learning with a Deep Algorithm

We now move on to discussing learning tasks.

1. The Semi-supervised Learning Task

Formally our goal is to solve the semi-supervised learning problem explained above. Practically, in order to solve this problem we will try to learn hidden variables $\Sigma_{|v|}$, $\Pi_{w,v}$ and $\sigma(v)$ (for v non-leaves) as well as the tree structure, and then use all of this information to determine labels. We begin with an easy claim.

Claim. *To identify the labels of the leaves in $Level(\ell) \setminus S$ correctly, then it must hold that for every a that appears in the leaves, there exists some $u, v \in S$ such that $a \in L(u) \cap L(v)$ and $MCA(\{\text{Leaves labeled by } a\}) = MCA(\{u, v\})$ where MCA denotes the most common ancestor. We refer to this as a good labeling .*

For example, if we only have one labelled example of a tiger there is no way to infer that some unlabeled examples are tigers, since there is no way to even figure out if the direct parent of the labeled example is also labeled as a tiger.

Proof. Observe that this is true even if we are given the tree, and being given the tree σ tells us nothing about the label assignments e.g. the label assignments and σ could have been both generated independently at random.

On the other hand, if the tree is known and we have a good labeling then it is obvious that we can infer all of the correct labels: first propagate the labels up the tree from S using the good labeling condition, then propagate this information down to the leaves (since labels are hereditary).

■

2. Recovering the Tree

What remains to discuss is the problem of recovering the tree. We will not attempt to use the labels for this task, so this part is an unsupervised learning task.

First we assume Σ is known. We consider the task of learning siblings.

Learning Siblings: First we consider how to test whether u, v are siblings with parent w given that $\Pi_{w,u} = \Pi_{w,v} = Id$. Suppose this is the case, then if we let

$$X := \#\{i \in [k] : \sigma(v)_i = \sigma(u)_i\}$$

we observe that there is a θ^2 chance of a given coordinate being "copied" from v to u and otherwise a $1/q$ chance of the coordinate being the same, thus

$$\mathbb{E}[X] = k\theta^2 + \frac{k(1-\theta^2)}{q}$$

and by the Chernoff bound this is very concentrated (sub-Gaussian with variance proxy of order k). Moreover if w and v are not siblings then the expectation is smaller, i.e. the first term is at least as bad as $k\theta^4$ so for sufficiently large k we can distinguish these two cases with high probability.

In general, we can do this test by enumerating over all possible $\Pi_{w,u}$ and $\Pi_{w,v}$: if w and v are not siblings, then with high probability we will never observe a small X, whereas if they are then when we choose the right permutations we will observe a small X with high probability.

Remark. Note that it is actually impossible to recover $\Pi_{w,v}$ and $\Pi_{w,u}$. If we take some permutation π and replace $\Pi_{w,v}$ with $\Pi_{w,v} \circ \pi$ and $\Pi_{w,u}$ with $\Pi_{w,u} \circ \pi$ then these new permutations will be equally good for the purposes of our statistical test above, and in fact if we make this change for a whole level we can see that the distribution of the generated data is identical, so the permutations Π are undeterdetermined. A technical argument can be used to show that this is the only degree of freedom for the Π and so we can recover up to this degree of freedom.

Recursive Algorithm: Now that we can learn siblings, we can recurse similarly to the case of tree reconstruction. Thus we can recover the tree and under the condition of a good labeling we can perform semi-supervised learning with $k = \theta(\ell)$.

We end by noting that if S is very large then there are other ways to learn labels without reconstructing the tree; e.g. if S consists of all but a few labels, then we can learn labels of unlabeled leaves just by finding their nearest-neighbor among the elements of S.

Lower Bounds

Note that in the above result we used a recursive ("deep") algorithm. This leads us to this natural question: can we show that simpler algorithms don't perform as well?

1. Local Algorithms

First we will show that local algorithms require a suboptimal number of samples. The definition of a local algorithm is that when it labels $w \notin S$, it uses only $\{(\sigma(v), L(v)) : v \in S\}$ and $\sigma(w)$; it does not use any of the other unlabelled data.

We choose the following input distribution over labels: Fix an integer h and $h_0 < h$. There are no labels for the first $h_0 - 1$ levels. At level h_0, we label each vertex distinctly, in a random order. For each vertex v at level h_0, we pick its leftmost and rightmost descendant (i.e. leftmost leaf of the overall tree under v and righmost leaf of the tree under v) and add these two vertices to S.

Theorem. *The probability that a local algorithm labels a random leaf correctly is less than* $d^{-h_0}(1 + O(k\theta^{h-h_0}))$.

Remark. Note that it is trivial to achieve d^{-h_0} by guessing randomly. As a consequence, to get better than random even when $h_0 = 1$ we need $k\theta^{h-1} = \Omega(1)$ so we need $k = \Omega(\theta^{-(h-1)}) = \Omega(|T|^{\beta})$ for some constant β depending on θ. In comparison the "deep" algorithm we described before needs only $k = \Omega(h) = \Omega(\log |T|)$.

Proof. [Sketch] The little cheat we make is that if w is in the same tree with labeled leaves v_1, v_2 then we assume $MCA(w, v_1) = MCA(w, v_2) = MCA(v_1, v_2)$ (It is in fact true that most v_1, v_2 are far apart, which is a general and easy to prove fact about trees that we can use to make the following argument rigorous).

Now the proof "trick" is to give the algorithm extra information, $(Label(v), \sigma(v))$ for each vertex v at level h_0. Given this extra information, $\sigma(w)$ is independent of $\{\sigma(v) : v \in S\}$ by the Markov property (since removing the vertices at level h_0 leaves w in a connected component by itself, relying on the "cheat" described above). We also make the algorithm's life "easy" by setting $\Pi = Id$ and $\Sigma = Id$.

Now we have reduced the problem to a pure hypothesis testing question. Let H_i be the hypothesis that w is labelled by i, i.e. w is a child of v_i where v_i is the vertex at level h_0 labeled by i. Note that the distribution of w given the true v_i is

that of a noisy copy with parameter θ^{h-h_0}. Thus with probability at least $1-k\theta^{h-h_0}$ all of the coordinates of w have been resampled from the uniform distribution. By a coupling argument, it follows that any hypothesis test labels correctly with probability at most

$$(1-k\theta^{h-h_0})(1/d) + k\theta^{h-h_0} \leq (1/d) + k\theta^{h-h_0}.$$

■

2. Shallow Algorithms

Next we want to show that certain shallow algorithms also cannot recover as well as the "deep" algorithm we presented first. Formally, we define a shallow algorithm to be one that is given access only the following data:

1. For each label a and each position i is given

$$c_i(a, x) = \#\{\text{leaves } \nu \text{ in } S \text{ labelled by } a \text{ such that } \sigma(v)_i = x\}$$

2. all unlabeled data

In other words, we have summarized the labeled data by count statistics and do not let the algorithm look at higher order moments of the labelled data.

We consider the family of input instances where the direct children of the root are labeled $1, \ldots, d$ and in each of their subtrees, every leaf is labeled except the rightmost child. Then we have the following theorem:

Theorem. *If $d\theta^2 < 1$, a shallow algorithm requires $k \geq (1/\theta)^c \geq \Omega(|T|^{c'})$ samples in order to label correctly with probability greater than $2/d$. This is true even if $\Sigma = Id$ and $\Pi = Id$.*

We do not present the whole proof of the theorem but just observe that it follows from known lower bounds on count reconstruction.

Finally we present an open problem: The lower bounds should be much better when Π are random and the Σ_j mix different elements. More specifically, the bound for shallow algorithms should hold if $\theta = 1-c^h$ for some $c < 1$ 'for some $c < 1$, instead of requiring $\theta < 1/\sqrt{d}$ when Π are random and Σ "mixes" well (e.g. $\Sigma_r(x) = x + 2^r$).

elmos / July 6, 2017

Mathematical Aspects of Deep Learning / Proudly powered by WordPress