Ch 20 Neural Networks: optimization

Friday, March 27, 2020 3:05 PM

III Optimization Error (how to train an ANN)

i.e., the catch!

TL;DR: bad news

Even with 1 hidden layer of 4 neurons, it's NP-hard to implement the ERM

Thm 20.7 let $k \geq 3$, $(\forall n)$ let (V,E) be a layered graph w/ n input nodes, $k+1$ nodes at single hidden layer, single output node. Then it is NP-hard to implement the ERM rule with respect to $\mathcal{H}_{V,E,\sigma=\text{sign}}$.

proof sketch

See exercise 8.3 part 2: reduce it to K-coloring a graph, known to be NP-Hard for $k \geq 3$

We're being loose, since it's not a decision problem

ex: Is 17 prime?

More important than NP-Hardness often is whether it is NP-hard to approximate

... but this is also true: it is NP-Hard to approx the ERM (Bartlett + Ben-David '02)

Making larger networks may help?
Try other activation functions?
} Consensus is "no"

So, what can we do?

1) Try a heuristic to solve ERM (eg SGD — but no guarantees now because it's nonconvex)

and hope it works

2) Explaining success of SGD,

A) usually avoids saddle-pts, and sometimes (wide networks) most/all local min are nearly as good as global min.

B) Switch to algorithmic-dependent guarantees (this also helps improve on VCdim generalization)

Both approaches very active as of 2020, "hot topics"

SGD: Stochastic: $\hat{L}_S(w) = \frac{1}{m}\sum_{i\in[m]} \ell(w,z_i)$

approximate with stochastic sampling, (new sample every iteration)

(ie., minibatch: $\mathbb{E}\ \frac{1}{m'}\sum_{i=1}^{m'} \ell(w,z_i) = \hat{L}_S(w)$

↗ i chosen iid w/ replacement uniform on $[m]$

gradient
(not even subgradient, since not convex)

Sign, ReLU not differentiable but that's just ignored

Compute automatically via Backpropagation
(= reverse-mode $\underbrace{\text{automatic differentiation}}_{\text{AD}}$)

i.e., automatically apply chain rule

Can do by hand, but usually in software package like
TensorFlow (that's a big reason to use an existing framework and not build your own)

If you can evaluate $h \in \mathcal{H}_{V,E,\sigma}$ in s seconds, $h: \mathbb{R}^n \to \mathbb{R}$,
then reverse-mode AD can compute $\nabla h \in \mathbb{R}^n$ in $\approx 4s$ seconds!
(but... possible memory explosion since must save intermediate calculations...
for large nets on a GPU, this is a big deal)

(forward-mode AD is efficient for functions $f: \mathbb{R} \to \mathbb{R}^k$)

Backprop "rediscovered" by ML people in '86, main ideas in '60s,
Seppo Linnainmaa 1976 basically the first, many Numerical Analysis papers in 80's (Griewank, one of them, describes history in 2012 article)