Approximating Hessians for Neural Networks

Maciej Skorski

University of Luxembourg

May 24, 2021

Outline

Optimization Landscape of Neural Networks

- *curvature* is critical for training...
  - (a) hessian controls accuracy of linearization (Taylor's formula)
  - (b) preconditions step and convergence (even for the simplest `SGD`!)
  - (c) used to boost convergence of Gradient Descents (`AdaHess`...)
  - (d) used to initialize MC (`NUTS`)
- ... but hard to compute in high dimension, e.g. neural networks (*billions* and more of hessian entries)

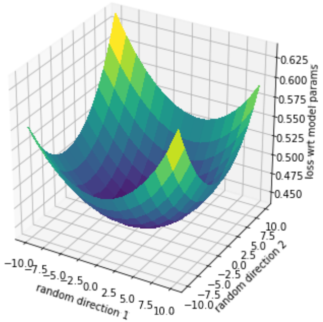

Figure 1: `LeNet5` loss (`FashionMNIST`). Visualization of the dependency on $> 44k$ parameters is done along two orthogonal directions chosen randomly.

Hessian Complexity?

- Non-trivial back-propagation, different than for gradients ☹

  $$D^2[f \circ g] = D^2 f \bullet Dg \bullet Dg + Df \bullet D^2 g,$$

  where $\circ$ and $\bullet$ denote, resp., composition and tensordot.
- Memory-consuming and in general infeasible ☹
- Limited support in some of AD frameworks (`JAX` and `TensorFlow` support both forward and reverse mode but `PyTorch` not (cf docs) ☹

Alternatives and Proposed Solution

- Compute HVPs (known in research, somewhat painful in practice)
  a. theoretically fast ☺
  b. limited in scope (vector products only) ☹
  c. issues with AD-software support ☹
- Hutchinson's Trick (`PyHess` and others):
  a. AD-software agnostic ☺
  b. limited in scope (diagonal only) ☹
  c. approximate and slow (probabilistic guarantees) ☹
- **Approximate Chain Rule** (this work)
  a. agnostic of AD-software ☺
  b. very fast ☺
  c. unlimited in scope (all hessian parts) ☺
  d. approximate, but works well for practical NNs! ☺

Approximate Chain Rule

- Let ℓ be the loss function, w be the parameters and $z(w)$ be the network output. Then:

$$D_w^2 \ell(z(w)) = \underbrace{D_z^2 \ell(z) \bullet D_w z(w) \bullet D_w z(w)}_{\text{linearization effect}} + \underbrace{D_z \ell(z) \bullet D_w^2 z(w)}_{\text{curvature effect}}.$$

- For typical networks the curvature effect is of smaller order!
- We claim that with good accuracy

$$D_w^2 \ell(z(w)) \approx \underbrace{D_z^2 \ell(z) \bullet D_w z(w) \bullet D_w z(w)}_{\text{linearization effect}},$$

  when *activations are nearly linear around 0* (satisfied in practice).
- The approximation costs just one back-propagation and two tensordots (hessian of loss wrt its natural domain is analytic) ☺

Local near-linearity of activations

- Approximation works for activations h s.t. $h(x) = \Theta(x) + O(x^3)$ for small x
- Satisfied for popular activations: `linear`,`tanh`,`sigmoid`,`ReLU`

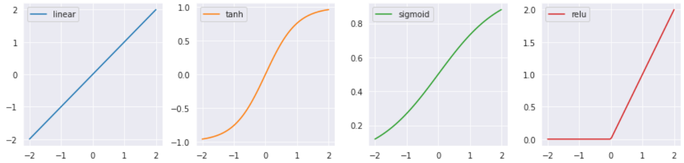

Figure 2: Behavior of activations around 0: second Taylor's term vanishes!

- This allow us to prove that the "curvature contribution" is negligible

Chain Rule Evaluation

- error term is of smaller order (theoretical guarantees)
- *no error* with variations of `ReLU`
- *small error* on popular nets: `LeNet`, `EfficientNet`, `ResNet`...

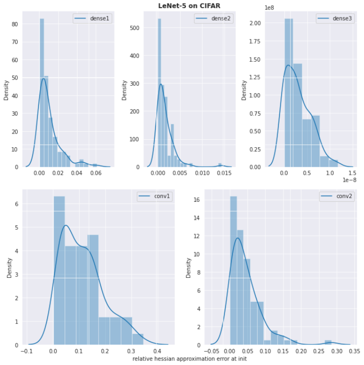

Figure 3: Approximation error at init. Main hessian components are evaluated on gradients.

Applications

- Quick approximate evaluation of any hessian components
- Speeding-up hessian-based optimizers
- Proofs for init schemes (replacing "variance-flow" heuristics)
- Improving convergence of specific nets, see the case study below ☺

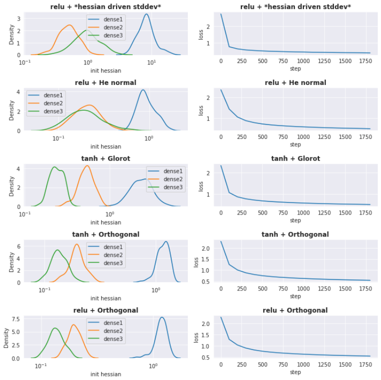

Figure 4: Network with layers of 128, 84 and 10 units on FashionMNIST.
The hessian-driven initialization outperforms established initializers.

Summing Up...

- We have a fast and accurate chain rule for hessian of NNs
- The idea is to neglect "curvature effects" appearing due to nearly-linear activation functions
- The approximation tested on a range of architectures
- Code at `https://github.com/maciejskorski/ml_examples`!

Working Philosophy ☺

- *learn diversity* of techniques (calculus, linear algebra, statistics)
- *divide and conquer*: attack problems with established tools
- *bridge to applications*: abstract findings, link to other areas...

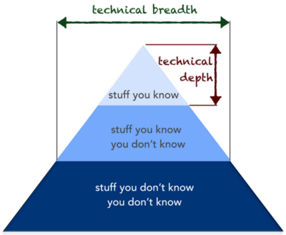
technical breadth
technical depth
stuff you know
stuff you know you don't know
stuff you don't know you don't know

Figure 5: Love this diagram!