Lec 13 Debugging & Interpretating NLP Models

Debugging

implementation wrong or model bad?

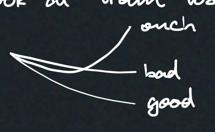
Situation: code of NN looks good, bad accuracy

Debug impl

- Everything's hparam
- Stochastic optim doesn't guarantee convergence

▷ Train time problems
- Model capacity lacking
- Bad training alg
- Train time bug

→ Look at train loss

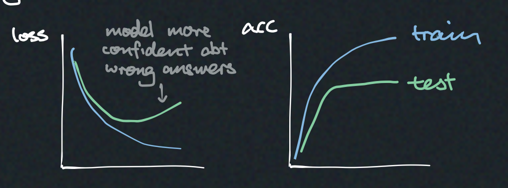


maybe by lottery ticket hypotesis?
↓
→ Model too small ← larger model can actually learn in fewer steps
→ Optimizer — try Adam?
→ LR — see other paper use
→ Initialisations?
→ Larger batches?

▷ Test time
- Train-test disconnect
  → Check eval vs train code for duplicate
  → ASSERT (batch loss = $\sum$ example loss)
- Search alg failing
  → ASSERT (loss on generated = decode score)

▷ Overfit

▷ Mismatch btwn optimised func & eval
e.g. loss, acc uncorrelated



e.g. model score and BLUE can uncorrelate
 └ beam ↑ ⇒ model likes shorter outputs
→ Reinforcement
→ Early stopping

Actionable eval

▷ Look at data
→ Something obviously off by one?
→ Any obvious things model's bad at?

Systematically:
→ Look at 100-200 errors, do typology
→ Use Zeno

Quantitative
→ After fix, if model better on that thing?

Interprep predictions

▷ Probing

Take big transformer model, cut open, attach probe (fix model, train probe) to test if some info is available at that point

BERT rediscover pipeline: earliers allow probe better at token stuff, later layers has more relation, ...

Problems:
- did probe solve the task by itself?
- maybe the probe's bad the model isn't?
- correlative, no causation
- data to train probe is bad?

* Interpret weights & activations?

Mechanistic inter.bility: reverse engineer NNs from their weights into understandable algorithmic units (called circuit)

▷ Weight
→ Edit them & see what happens
 └ Yap just poke at it

Model editing
- Target: change specific fact the model knows
- Approach: change some weight st. only that changes

e.g. Graham is prof at ~~CMU~~ Stanford

→ Causal tracing to find what weight to change

▷ Activations

* Steering vector: fixed-len vec that steers LM to output sth specific when added to activation at an exact place

The steering vec's learnt, we can decide where to inject
In general: first time step, inject at middle of model

Usually we can find steering vec for most sequences
Even short random steering seqs can have steering vec

They are interpretable
- Distance correlates to semantic distance
- Style transfer by vector arithmetics
- Interpolation often works!

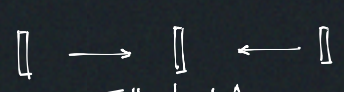
▷ Inference-time Intervention

Use linear probe, find attention that corresponds with sth
Then shift attention head activation

▷ Contrastive Steering Vec

Use contrastive prompt on subject A to get neutralised steering vec that steers towards subject A

e.g. I love A I hate A



Talk about A