Western Science

Artificial Intelligence II

Part 2: Lecture 10

Yalda Mohsenzadeh

Advice on Deep Learning

Today

- Tour of collective advice

- Q&A about deep learning in general and your projects in particular

Experience and Expertise

- Skill or knowledge
- knowing most of mistakes that can be made in a particular field

Data

know your data! if you don't, how will you know if your model does?

Data

- inspect the distribution of the inputs and targets
  - inspect random selection of inputs and targets to have a general sense
  - histogram input dimensions to see range and variability
  - histogram targets to see range and imbalance
  - select, sort, and inspect by type of target or whatever else

Data

- inspect the inliers, outliers, and neighbors
  - visualize distribution and outliers, especially outliers, to uncover dataset issues
  - look at **nearest neighbors** in the raw data, or pre-trained net for same domain, or randomly initialized net
  - examples: rare grayscale images in color dataset, huge images that should have been rescaled, corrupted class labels that had been cast to uint8

Data

- pre-processing: the data as it is loaded is not always the data as it is stored!
  - inspect the data as it is given to the model by **output = model(data)**

Data

- pre-processing:
  - **summary statistics:** check the min/max and mean/variance to catch mistakes like forgetting to standardize
  - **shape:** are you certain of each dimension and its size? sanity check with dummy data of prime dimensions: there are no common factors, so mistaken reshaping/flattening/permuting will be more obvious. example: a 64x64x64x64 array can be permuted without knowing.
  - **type:** check for casting, especially to lower precision? how does standardization change integer data?

Data

- Pre-processing:
  - start with less, and then add more… once everything works at all
  - for instance: just standardize at first, and only augment once there is a model and it fits

Data

- resample the data to decorrelate: that is, remember to shuffle
- consider selecting miniature train and test sets for development and testing: these should be chosen once and fixed throughout optimization + evaluation
- it's heartbreaking to wait an entire epoch and then and only then have your experiment-to-be crash

Data

- check if you can do the task, as it is given to the model

    - (consider the task with and without pre-processing)

- ...if it's a reasonable perceptual task for a human

Model

keep it as simple as you can!

Model

- keep it as simple as you can!
  - sure, there are sophisticated models out there
  - but they were made by either
    - (1) going step-by-step, from simple to complex or
    - (2) suffering, madness, and the gnashing of teeth

Model

- keep it as simple as you can!
  - do your first experiment with the simplest possible model w/ and w/o your idea
  - at least you will be armed with a little understanding
  - then follow-up with bidirectional search between simple models and state-of-the-art models with your own model in the middle

Model

- **resist snowflake syndrome!** try defaults first

  but my data/task/idea is special

  ...no it's not, or not until proven so (by say defeating a residual net)

- You may replace 1 month+ of model development with LeNet and it end up being more accurate and >10,000x faster

- even if an off-the-shelf net does not prevail, it can serve as a baseline

Model

- simple baselines catch many issues and swiftly too

- learn a linear model on random features by fixing all of the parameters at their initialization except for the output

- zero out the data by x.zero_() and check that results are worse

- double-check the model actually is what you thought you defined

Optimization

we are still learning how to optimize deep nets

Optimization

- we are still learning how to optimize deep nets

- much progress is being made but it's nevertheless a dark forest

- explore if you like, but balance exploration by exploitation of a known good setting, or the closest setting you can find

Optimization

- figure out optimization on **one/few/many points** in that order
  - overfit to a single data point
  - then fit a batch
  - and finally try fitting the dataset (or a miniature edition of it)
- to discover issues as soon as possible

Optimization

- sanity check the loss against a suitable reference value
  - classification with cross-entropy loss: uniform distribution
  - regression with squared loss: mean of targets (or even just zero)
- and if your loss is constant, double-check for zero initialization of the weights

Optimization

- the learning rate and batch size are more-or-less the cardinal hyperparameters
  - choose the learning rate on a small set (see "Stochastic Gradient Descent Tricks" by Leon Bottou)
  - simplify your life and **use a constant rate**, that is, no schedule until everything else is figured out
  - schedule according to epochs (== number of passes through the data), not number of iterations

Optimization

- clear gradients after each iteration or else they accumulate
- remember opt.zero_grad()!
- remarkably, with adaptive optimizers and enough time a model can still learn if the gradients are never cleared out… but it's really sensitive

Optimization

- live on the edge and try extreme settings (but just a little bit)
- If optimization never diverges, your learning rate is too low.

- If you’ve never missed a flight, you’re spending too much time in airports.

Optimization

- check the sign of the loss
- surely ascent is not descent, but that can be hard to remember in the moment
- definitely for custom losses! the defaults were checked for you, not so your own

Evaluation

switch to evaluation mode by model.eval()

Evaluation

- switch to evaluation mode by model.eval()
- no, really
- and check the mode by model.training

Evaluation

- know the output! metrics and summaries can obscure all kinds of issues
- inspect input-output pairs across min/max/quartiles of the loss and other metrics of interest
- keep an eye on the output and loss for a chosen set across iterations to have a sense of the learning dynamics (the chosen set could be the whole validation set)
- doesn't hurt to eyeball a few subsets chosen at random in case you catch anything surprising

Evaluation

- separate evaluation from optimization!
- save checkpoints at intervals and evaluate them offline
- there are the perils of nondeterminism, batch norm, etc. when mixing training and testing
- plus it only slows down training

Evaluation

- evaluate on the whole set: batch-wise statistics, even if smoothed, are too noisy
- if this is slow, then it's all the more reason to decouple evaluation from training
- if the evaluation is truly massive, consider a miniature set for routine evaluation and only evaluate on the full set at longer intervals

Tuning

Try the scientific method

Tuning

- **try the scientific method:** change one thing at a time
- **not the computer scientific method:** change everything every time
- ...except perhaps when you really have no idea what's going on

Tuning

- random search over grid search

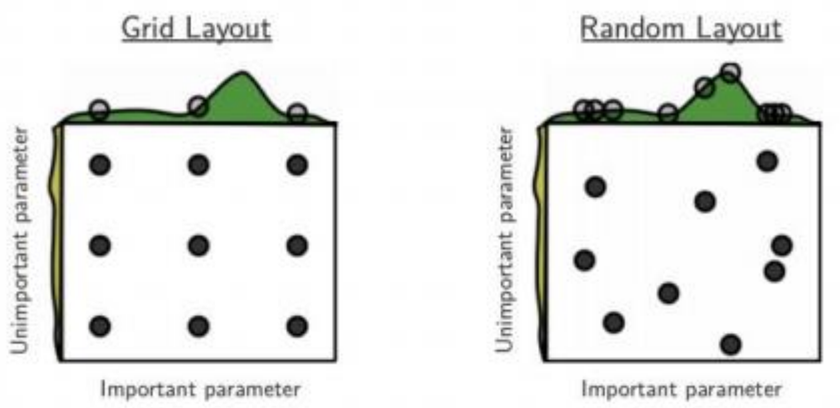

Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Bergstra & Benjio, JMLR 2012

Tuning

- Coarse-to-fine search for hyperparameters

- wider search for shorter training (on the same data: remember to set the seed!)

- narrower search for longer training

- but be careful with greed! still need to tune for full training, because the best hyperparameters for short runs may not prevail in the end

Testing

Testing

- test correctness or settle for incorrectness, as there's very little middle ground
- anything untested might be wrong (if not now, then later)
- rely on a separate, simpler implementation as a reference

Testing

- check gradients and do it thoroughly

- gradcheck is the checker bundled into PyTorch
- cs231n has gradient checking rules of thumb
- Tim Vieira has further tips for accurate and thorough checking

Computation

more hardware, more problems

Computation

- more hardware, more problems don't parallelize immediately
- make your model work on a single device first
- attempt to parallelize on a single machine
- only then go to a multi machine setup
- and check that iterations/time actually improves!

- see Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour for good advice

So, is Deep Learning a Piece of Cake?

- Not quite.
- The tools are better every day, but tools are no substitute for thought.
- While there are many layers, hopefully you're now more ready to cut through the complexity.
- Next time you bake, you could try Andrej Karpathy's recipe: http://karpathy.github.io/2019/04/25/recipe/