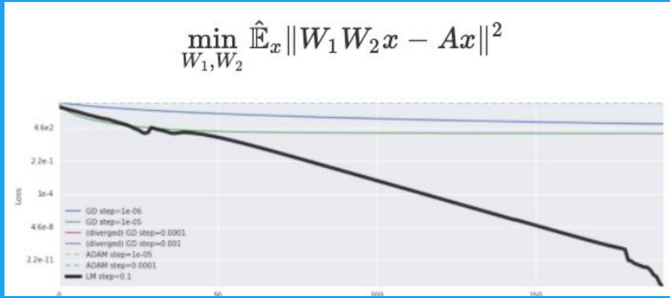
Harnessing second order optimizers from deep learning frameworks

Ryan Turner

April 16, 2021

Background

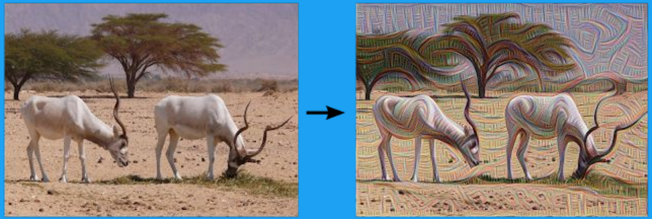
- Methods like L-BFGS and conjugate gradient (CG) were the go-to methods in the pre-deep learning era
  - e.g., minimize.m
- What changed in DL?
  - Huge data sets ⇒ exact gradients infeasible so we needed SGD
  - Luckily in DL tasks, SGD tends to work better anyway
- But, optimization is a broader problem than weights of a deep net
- In other problems with exact gradients, traditional optimization can work better
- Also, works without hyper-parameter tuning and tricks
- Better handling of constraints

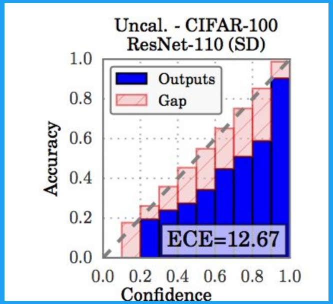


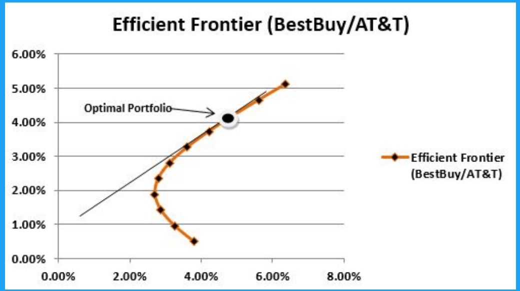
(from controversial NeurIPS 2017 Rahimi and Recht presentation)

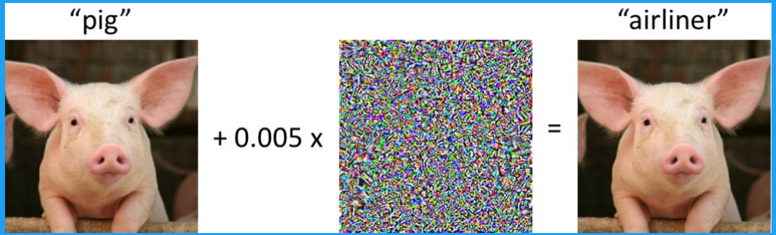
Examples

- Calibration layers
- Deep dream
- Adversarial examples
- General (OR-style) optimization
    - e.g., portfolio optimization
- etc









Motivation (Scenario)

- Have a cool objective function to optimize
  - No giant data ⇒ Nice, we don't need stochastic gradients
  - Derivatives are complex, several tensors to optimize ⇒ no problem, we have autodiff
  - Maybe, weird conditioning ⇒ no problem, we have second order optimizers
- But faced with 2 options:
  - Use autodiff package SGD (the new world):
    - Only simple gradient descent 😞
    - Must tune hyper-parameters 😞
    - Often requires tricks (grad-norm, grad-clip, ...) 😞
  - Or scipy optimize (the old world)
    - Many optimization options 😀
    - Principled support for constraints and bounds 😀
    - Only optimizes vectors 😞
    - Doesn't play nice with autodiff packages 😞
      - Was built assuming you write gradients manually

In Python, go to is SciPy for 2nd order optimization

- But it expects a f(x), where x is `ndarray` of shape (n,)
- We like the style of TensorFlow/PyTorch optimization where x = {‘foo’: A, ‘bar’: B, …}
  a. And A and B are arbitrary shaped tensors
- Scipy requires:
  a. Packing into a vector
  b. Converting everything into numpy
  c. Even needs to be float64
- Becomes repetitive hassle
- So, dict-minimize *can take care of this for you*

scipy.optimize.minimize

scipy.optimize.minimize(*fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None*) [source]

Minimization of scalar function of one or more variables.

Parameters: fun : *callable*

The objective function to be minimized.

```
fun(x, *args) -> float
```

where `x` is an 1-D array with shape (n,) and `args` is a tuple of the fixed parameters needed to completely specify the function.

x0 : *ndarray, shape (n,)*

Initial guess. Array of real elements of size (n,), where ‘n’ is the number of independent variables.

args : *tuple, optional*

Extra arguments passed to the objective function and its derivatives (*fun, jac* and *hess* functions).

method : *str or callable, optional*

Type of solver. Should be one of

- ‘Nelder-Mead’ (see here)
- ‘Powell’ (see here)
- ‘CG’ (see here)
- ‘BFGS’ (see here)
- ‘Newton-CG’ (see here)
- ‘L-BFGS-B’ (see here)
- ‘TNC’ (see here)
- ‘COBYLA’ (see here)
- ‘SLSQP’ (see here)
- ‘trust-constr’(see here)
- ‘dogleg’ (see here)
- ‘trust-ncg’ (see here)
- ‘trust-exact’ (see here)
- ‘trust-krylov’ (see here)
- custom - a callable object (added in version 0.14.0), see below for description.

If not given, chosen to be one of `BFGS`, `L-BFGS-B`, `SLSQP`, depending if the problem has constraints or bounds.

Dict-minimize

- Gives use the dictionary of parameters interface we want
- Provides interfaces too:
    - TensorFlow
    - PyTorch
    - JAX
    - NumPy
- Try it out whenever you can handle exact gradients
- Usually
    - no hyper-parameter tuning is required 😀
    - no gradient manipulation tricks 😀
- People have been conditioned to use SGD in all optimization use cases instead of where it is needed
    - Try out alternatives when you can

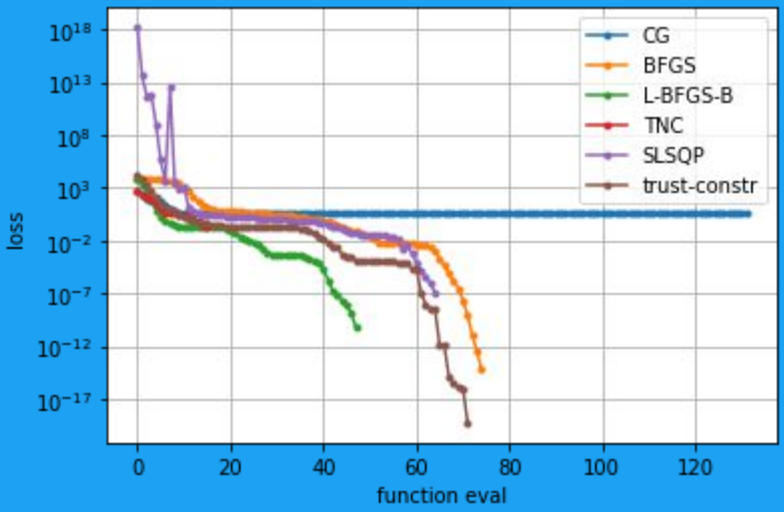
Want API simplicity

- Just replace:
```
from scipy.optimize import minimize
```
with
```
from dict_minimize.torch_api import minimize
```
or
```
from dict_minimize.tensorflow_api import minimize
```
or
```
from dict_minimize.jax_api import minimize
```
or
```
from dict_minimize.numpy_api import minimize
```
- Mirrors the original SciPy API

Now Examples!

Rosenbrock in all 4 frameworks here

- Rosenbrock is MNIST of optimization
- Show that all the built in algos work *without tricks*



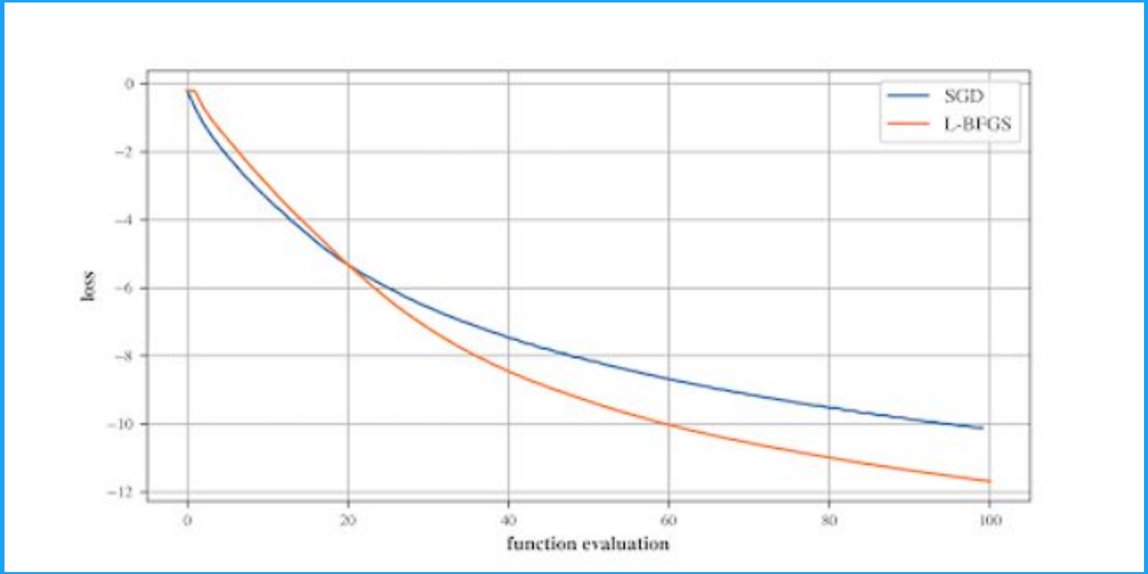
Apply to DeepDream

- Optimize wrt the input with weights fixed
  - ⇒ no giant dataset ⇒ we can get exact grad
- Visualize neurons in InceptionV3
  - Optimize input w.r.t. activations
- Mid-layers responds to texture
- Another advantage of dict-minimize:
  - Built in support for bounds (pixel must be in [0,1])
  - No clipping tricks
- No hyper-parameters to tune with L-BFGS
  - SGD version requires tricks (gradient norm and gradient clipping)
  - L-BFGS version: just use the actual gradient, no need to manipulate it with tricks
- Just works

Crocodile texture a lab 😮

[video]

Apply to DeepDream



pip install dict_minimize

GitHub | Read the Docs | PyPI | Blog

Thank you