ensmallen: a flexible C++ library for efficient function optimization

Shikhar Bhardwaj[1], Ryan R. Curtin[2], Marcus Edel[3], Yannis Mentekidis, Conrad Sanderson[4,5]

[1]Delhi Technological University, [2]RelationalAI, [3]Free University of Berlin, [4]Data61/CSIRO, [5]University of Queensland

Motivation

Mathematical **optimization is the workhorse of virtually all machine learning algorithms**. For a given objective function $f(\cdot)$, almost all machine learning problems can be boiled down to the following optimization form:

$$\operatorname*{argmin}_x f(x).$$

$\Rightarrow$ **ensmallen**, a C++ optimization toolkit that contains a wide variety of optimization techniques

Types of Objective Functions

- **ensmallen** provides a set of **optimizers** for optimizing user-defined objective functions
- **arbitrary**: no assumptions can be made on $f(x)$
- **differentiable**: $f(x)$ has a computable gradient $f'(x)$
- **separable**: $f(x)$ is a sum of individual components: $f(x) = \sum_i f_i(x)$
- **categorical**: x contains elements that can only take discrete values
- **sparse**: the gradient $f'(x)$ or $f_i'(x)$ (for a separable function) is sparse
- **partially differentiable**: the separable gradient $f_i'(x)$ is also separable for a different axis j
- **constrained**: x is limited in the values that it can take

Feature Comparison

	unified framework	constraints	batches	arbitrary functions	arbitrary optimizers	sparse gradients	categorical
ensmallen	●	●	●	●	●	●	●
Shogun	●	-	●	●	●	-	-
Vowpal Wabbit	-	-	●	-	-	-	●
TensorFlow	●	-	●	◐	-	◐	-
Caffe	●	-	●	◐	◐	-	-
Keras	●	-	●	◐	◐	-	-
scikit-learn	◐	-	◐	◐	-	-	-
SciPy	●	●	-	●	-	-	-
MATLAB	●	●	-	●	-	-	-
Julia (Optim.jl)	●	-	-	●	-	-	-

●= provides feature, ◐= partially provides feature, - = does not provide feature.

Runtime

Runtimes for the linear regression function on various dataset sizes (n = number of samples, d = dimensionality of each sample). 10 iterations of L-BFGS.

algorithm	d: 100, n: 1k	d: 100, n: 10k	d: 100, n: 100k	d: 1k, n: 100k
ensmallen-1	**0.001s**	**0.009s**	**0.154s**	**2.215s**
ensmallen-2	0.002s	0.016s	0.182s	2.522s
Optim.jl	0.006s	0.030s	0.337s	4.271s
scipy	0.003s	0.017s	0.202s	2.729s
bfgsmin	0.071s	0.859s	23.220s	2859.81s
ForwardDiff.jl	0.497s	1.159s	4.996s	603.106s
autograd	0.007s	0.026s	0.210s	2.673s

Optimization Algorithms

ensmallen provides a large set of diverse optimization algorithms for various objective functions:

- **SGD variants:** Stochastic Gradient Descent (SGD), SGD with Restarts, Parallel SGD (Hogwild!), Stochastic Coordinate Descent, AdaGrad, AdaDelta, RMSProp, SMORMS3, Adam, AdaMax, NadaMax, AMSGrad, Nadam, OptimisticAdam, WNGrad, EVE, FTML, pAdam, SWATS
- **Quasi-Newton variant:** Limited-memory BFGS (L-BFGS), incremental Quasi-Newton method, Augmented Lagrangian Method
- **Genetic variants:** Conventional Neuro-evolution, Covariance Matrix Adaptation Evolution Strategy, SPSA
- **Other:** Conditional Gradient Descent, Frank-Wolfe algorithm, Simulated Annealing

Interface

For the **most common case of a differentiable** $f(x)$**, the user only needs to implement two methods**:

- **double Evaluate(**x**)**: given coordinates x, this function returns the value of $f(x)$.
- **void Gradient(**x**,** g**)**: given coordinates x and a reference to g, set $g = f'(x)$.

or one function that computes **both** $f(x)$ **and** $f'(x)$ **simultaneously:**

- **double EvaluateWithGradient(**x**,** g**)**

Example - Linear Regression Function

Implementation of objective and gradient functions for linear regression, used by optimizers in **ensmallen**. The types `arma::mat` and `arma::vec` are matrix and vector types.

```
class LinearRegressionFunction {
 public:
  // Construct the LinearRegressionFunction with the given data.
  LinearRegressionFunction(arma::mat& X_in, arma::vec& y_in) : X(X_in), y(y_in) {}

  // Compute the objective function,
  double Evaluate(const arma::mat& theta) {
    return std::pow(arma::norm(y - X * theta), 2.0);
  }

  // Compute the gradient and store in 'gradient'.
  void Gradient(const arma::mat& theta, arma::mat& gradient) {
    gradient = -2 * X.t() * (y - X * theta);
  }

  // Compute the objective function and gradient store in 'gradient'.
  double EvaluateWithGradient(const arma::mat& theta, arma::mat& gradient) {
    const arma::vec v = (y - X * theta); // Cache result.
    gradient = -2 * X.t() * v;  // Store gradient in the provided matrix.
    return arma::accu(v % v); // Take squared norm of v.
  }
 private:
  arma::mat& X; arma::vec& y;
};
```

Example - Optimization

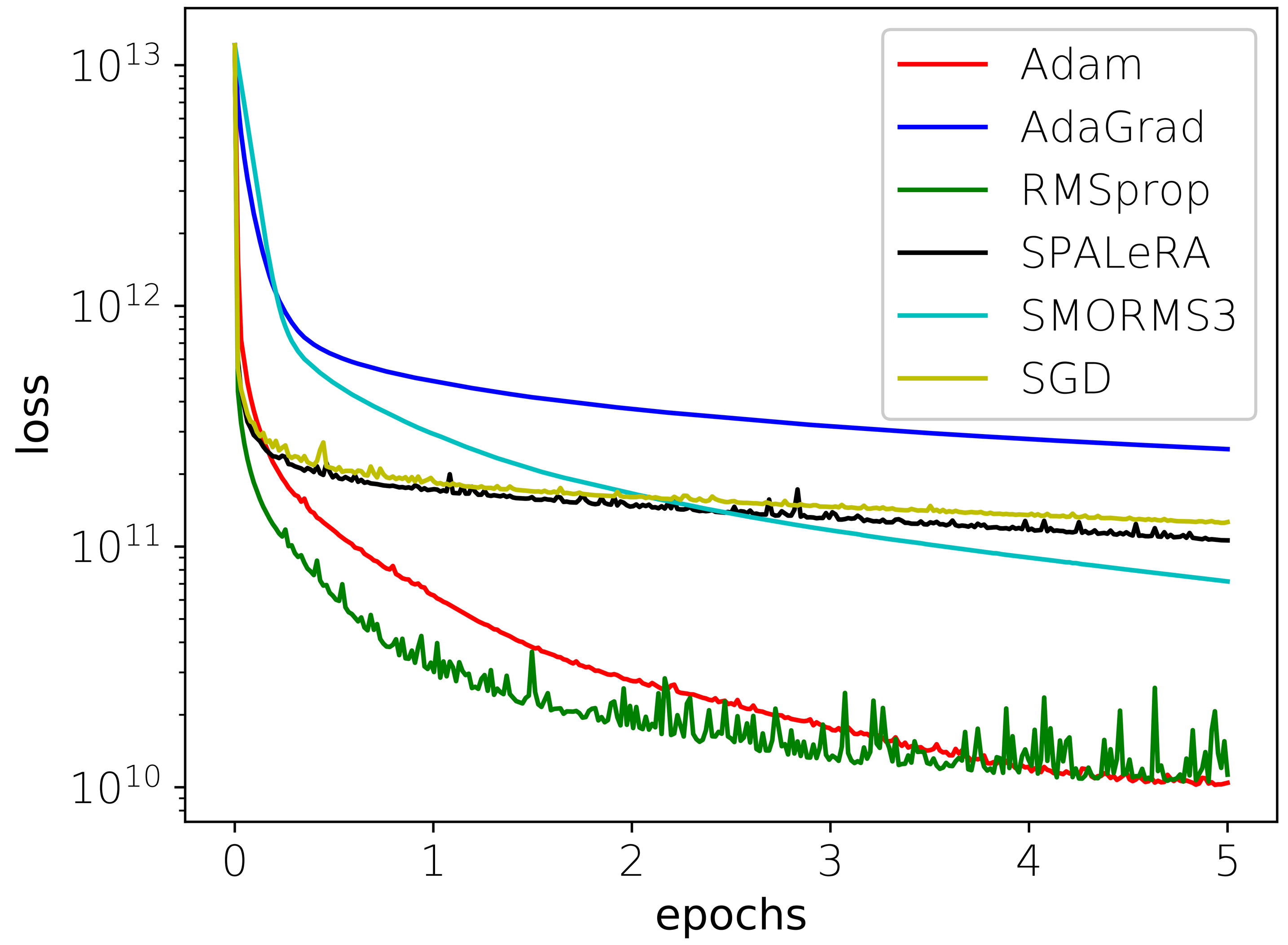
Given the defined `LinearRegressionFunction` class, find the best parameters θ:

```
LinearRegressionFunction lrf(X, y); // We assume X and y already hold data.

using namespace ens;

// After this call, the second parameter holds the solution.
L_BFGS().Optimize(lrf, lbfgsModel); // Use the BFGS to get solution.
StandardSGD().Optimize(lrf, sgdModel); // Use the SGD to get solution.
Adam().Optimize(lrf, adamModel); // Use Adam to get solution.
AdaGrad().Optimize(lrf, adagradModel); // Use AdaGrad to get solution.
SMORMS3().Optimize(lrf, smorms3Model); // Use SMORMS3 to get solution.
SPALeRASGD().Optimize(lrf, spaleraModel); // Use SPALeRASGD to get solution.
RMSProp().Optimize(lrf, rmspropModel); // Use RMSProp to get solution.
```

It's easy to plug in different optimizers and compare their performance!



Conclusions

- **ensmallen**, a flexible C++ library for function optimization
- provides an **easy interface** for the implementation and optimization
- supports separable and constrained functions
- provides **many pre-built optimizers** (including numerous variants of SGD and Quasi-Newton optimizers)
- **automatically generating missing methods** $\Rightarrow$ makes the implementation of objective functions easier

Project page http://ensmallen.org
Github http://github.com/mlpack/ensmallen