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: performance comparisons with other optimization libraries. can be boiled down to the following optimization form: $\frac{1}{200}$ is used for example $\frac{1}{200}$ is used for example $\frac{1}{200}$ is used for example $\frac{1}{200}$

 $argmin f(x)$. \overline{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. It is also easy to implement a new optimizer in the ensuing state in the ensuing \blacksquare an easy-to-use library that can solve that can solve that can solve that takes are more takes any function f(x) that takes are more takes any function f(x) that takes are more takes any function f(x) that takes are more ta

- ensmallen provides a set of optimizers for optimizing user-defined objective functions ϵ or ϵ or ϵ , ϵ or ϵ ontiminate ϵ is a that in ϵ is a defined shipping functure; **Fishianen** provides a set or **optimizers** for optimizing user-defined objective funcadvantage of this structure. For example, in addition to f(x), a user can provide an implementation to f(x), a u
	- arbitrary: no assumptions can be made on $f(x)$
- \bullet differentiable: $f(x)$ has a computable gradient $f'(x)$ to significant speedups.
- separable: $f(x)$ is a sum of individual components: $f(x) = \sum_i f_i(x)$
- \bullet categorical: x contains elements that can only take discrete values listed below:
	- sparse: the gradient $f'(x)$ or f'_i \bullet sparse: the gradient $f'(x)$ or $f'_i(x)$ (for a separable function) is sparse
	- \bullet partially differentiable: the separable gradient f_i' \bullet partially differentiable: the separable gradient $f_i'(x)$ is also separable for a different axis j $\overline{\text{axis } j}$
	- \bullet constrained: x is limited in the values that it can take

Shikhar Bhardwaj¹, Ryan R. Curtin², Marcus Edel³, Yannis Mentekidis, Conrad Sanderson^{4,5} 1 Delhi Technological University, 2 RelationalAI, 3 Free University of Berlin, 4 Data61/CSIRO, 5 University of Queensland batches of data points at a time. Other machine libraries, such a time libraries, such as sciences, such a tim \sim optimization algorithms but not in a coherent or reusable framework. Many programming languages \sim available and used for many decades. However, these implementations are often unsure often unsure of α

widtely used in the Python community, and Matters function of the Python Community, and Matters function \mathcal{L}

applied to. We discuss the mechanisms by which ensuing the mechanisms by which ensuing the mechanisms by which In this paper, we describe the functionality of ensmallen and the types of problems that it can be efficiency and ease-of-use. We show a few examples that use that use that use that use that use that use that u
In the library, as well as well as well as well as empirically as well as the library, as well as well as the

refers to if support for categorical features exists. samples, $d=$ dimensionality of each sample). 10 iterations of L-BFGS. Runtimes for the linear regression function on various dataset sizes ($n=$ number of

 \bullet = provides feature, \bullet = partially provides feature, - = does not provide feature. \bullet = provides feature, \bullet = partially provides feature, - = does not provide feature.

d: 1k, *n*: 100k 603.106s

- 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, WN-Grad, 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

batches indicate support for constrained problems and batches; *arbitrary functions* means arbitrary objective

functions are easily implemented; *arbitrary optimizers* means arbitrary optimizers are easily implemented; Runtime *sparse gradient* indicates that the framework can natively take advantage of sparse gradients; and *categorical*

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)

samples, and d indicating the dimensionality of each sample. All Julia runs do not count compilation time.

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

- \bullet ensmallen, a flexible $C++$ library for function optimization
-
- supports separable and constrained functions
- Quasi-Newton optimizers)
- objective functions easier

Optimization Algorithms

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

Interface

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!

• provides an easy interface for the implementation and optimization

• provides many pre-built optimizers (including numerous variants of SGD and

• automatically generating missing methods \Rightarrow makes the implementation of

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

Conclusions