Stochastic Gradient Descent in Python

Subhayan De, Ph.D.

1 Stochastic Gradient Descent Module

Download the SGD module from https://github.com/CU-UQ/SGD. See the demo (`sgd_demo.py`) for an example of the implementation.
For a description of the algorithms see Ruder (2010).
Required packages: numpy, time
NOTE: Currently, the stopping conditions are maximum number of iteration and 2nd norm of gradient vector. Only time-based and exponential learning schedules are implemented.

Report any bugs to Subhayan.De@colorado.edu

2 Algorithms implemented

A module named `StochasticGradientDescent` has been created where as of February, 2019, twelve different stochastic gradient descent algorithms have been implemented, namely,

- stochastic gradient descent (class name `SGD`)
  - with momentum
  - with Nesterov momentum (NAG)
- mini-batch gradient descent (class name `minibatchSGD`)
- RMSprop (class name `RMSprop`)
- AdaGrad (class name `AdaGrad`)
- Adam (class name `Adam`)
- Adamax (class name `Adamax`)
- Nesterov accelerated Adam (class name `Nadam`)
- Adadelta (class name `Adadelta`)
- Stochastic average gradient (class name `SAG`)
- Stochastic variance reduced gradient descent (class name `SVRG`)

2.1 SGD class:

```
================================================================================
|                      Stochastic Gradient Descent class                       |
================================================================================
Initialization:
sgd = SGD(obj, grad, eta, param, iter, maxIter, objFun, gradFun,
lowerBound, upperBound, stopGrad, momentum, nesterov,
learnSched, lrParam)

NOTE: To perform just one iteration provide either grad or graduFn.
obj  or objFun are optional.
================================================================================
Attributes:
obj:        objective (optional input)
grad:       Gradient information
(array of dimension nParam-by-1, optional input)
eta:        learning rate ( = 1.0, default)
param:      the parameter vector (array of dimension nParam-by-1)
nParam:     number of parameters
iter:       iteration number
maxIter:    maximum iteration number (optional, default = 1)
objFun:     function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:    function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound: lower bound for the parameters (optional input)
upperBound: upper bound for the parameters (optional input)
paramHist:  parameter evolution history
stopGrad:   stopping criterion based on 2-norm of gradient vector
momentum:   momentum parameter (default = 0)
nesterov:   set to True if Nesterov momentum equation to be used
(default = False)
learnSched: learning schedule (constant, exponential or time-based,
default = constant)
lrParam:    learning schedule parameter (default =0.1)
alg:        algorithm used
__version__:version of the code
================================================================================
Methods:
Public:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
update:         perform a single iteration
```

```
performIter:    perform maxIter number of iterations
getParamHist:   returns parameter update history
Private:
__init___:          initialization
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Bottou, Léon, Frank E. Curtis, and Jorge Nocedal.
"Optimization methods for large-scale machine learning."
SIAM Review 60.2 (2018): 223-311.
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.1.1 Example

NOTE: Implementation of this example is in:

`sgd_demo.py`

Consider the following linear regression problem:

$$y = 3 + 4.5x + \text{noise} \tag{1}$$

where the regression parameters are $\theta = [3, 4.5]^T$.

Using 1000 measurements and an initial guess of of $\theta_0 = [2, 0.5]^T$ the above algorithms are implemented and run for 2500 iterations and a stopping criterion for 2nd norm gradient to be less than 10^{-6}.

Objective function is provided in `objFun` and gradient function is provided in `gradFun`. The problem is initialized using

```
# initial parameter
w10 = 2.0
w20 = 0.5
theta = np.array([w10, w20])
R = objFun(theta) # initial objective
it = 0 # set iteration counter to 0
maxIt = 2500 # maximum iteration
dR = gradFun(theta) # initial gradient
```

2.1.2 `SGD` class:

For the vanilla stochastic gradient descent:

```
import SGD as sgd

# Stochastic Gradient Descent
eta = 0.0025 # learning rate

opt = sgd.SGD(obj = R, grad = dR, eta = eta, param = theta, iter = it,
maxiter=maxIt, objFun=objFun, gradFun=gradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist() # get parameter update history
```

The output is be the following:

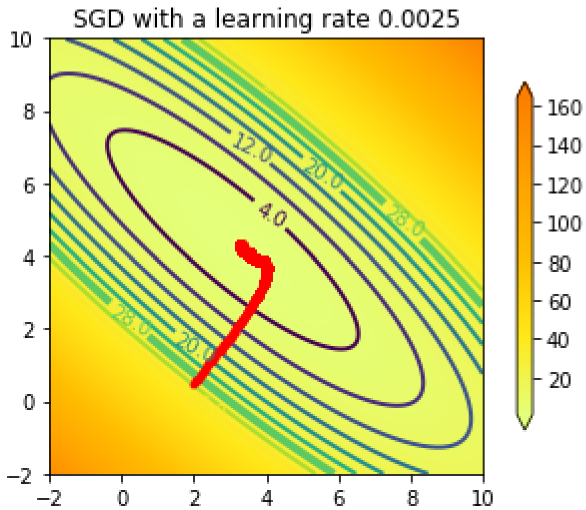
```
Learning rate =  0.0025

Learning schedule:  time-based

Algorithm:  SGD

SGD |██████████████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 77.22s, Objective = 0.943425
```

If we plot the parameter history:



2.1.3 SGD with `time-based` learning schedule:

```
# Stochastic Gradient Descent
eta = 0.1 # learning rate

opt = sgd.SGD(obj = R, grad = dR, eta = eta, param = theta, iter = it,
```

```
maxiter=maxIt, objFun=objFun, gradFun=gradFun,
learnSched = 'time-based', lrParam =0.5) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist() # get parameter update history
```

The output is be the following:

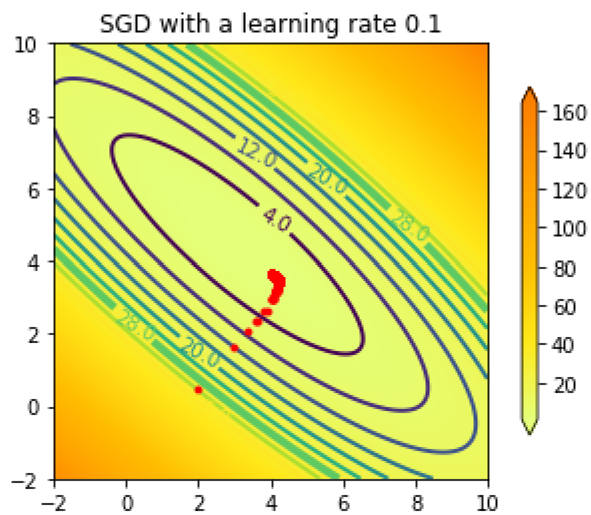
```
Learning rate =  0.1

Learning schedule:  time-based

Algorithm:  SGD

SGD |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 77.5s, Objective = 1.166173
```

If we plot the parameter history:



2.1.4 SGD + momentum:

```
eta = 0.001 # learning rate

opt = sgd.SGD(obj = R, grad = dR, eta = eta, param = theta, iter = it,
maxiter=maxIt, objFun=objFun, gradFun=gradFun,
momentum = 0.9) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

The output is the following:

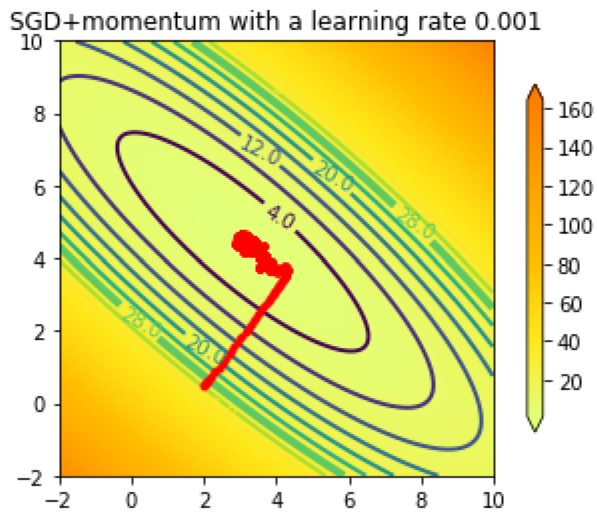
```
Learning rate =  0.001

Learning schedule:  constant

Algorithm:  SGD+momentum

SGD+momentum |██████████████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 77.52s, Objective = 0.94187
```

If we plot the parameter history:



2.1.5 SGD + Nesterov momentum:

```
eta = 0.001 # learning rate

opt = sgd.SGD(obj = R, grad = dR, eta = eta, param = theta, iter = it,
maxiter=maxIt, objFun=objFun, gradFun=gradFun,
momentum = 0.9, nesterov = True) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

The output is the following:

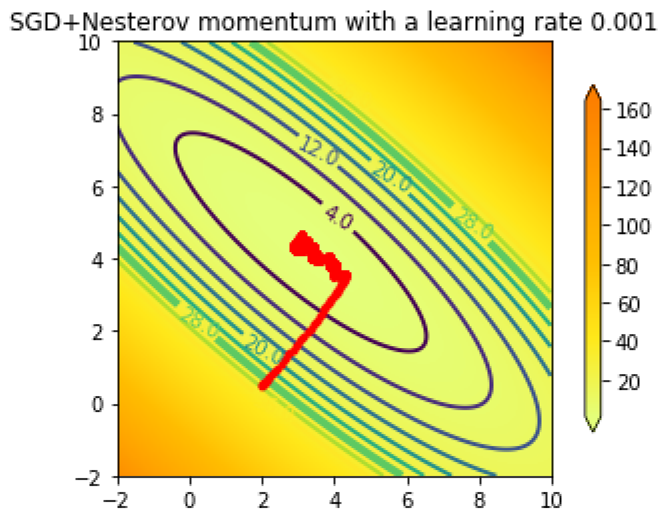
```
Learning rate =  0.001

Learning schedule:  constant
```

```
Algorithm:  SGD+Nesterov momentum

SGD+Nesterov momentum |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 116.86s, Objective = 0.965793
```

If we plot the parameter history:



2.2 minibatchSGD

```
==============================================================================
|                              minibatch SGD class                            |
|                 derived class from Stochastic Gradient Descent               |
==============================================================================
Initialization:
mbsgd = minibatchSGD(nSamples, nTotSamples,newGrad = 0.0,
obj, grad, eta, param, iter, maxiter,
objFun, gradFun, lowerBound, upperBound)

==============================================================================
Attributes:
alg:            minibatchSGD
eta:            learning rate
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
newGrad:        gradient information
(array of dimension nParam-by-nSamples)
nSamples:       number of gradients updated at each iteration
iter:           iteration number (optional)
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
```

```
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optionalinput)
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
learnSched:     learning schedule (constant, exponential or time-based,
default = constant)
lrParam:        learning schedule parameter (default =0.1)
alg:            algorithm used
__version__:    version of the code
==============================================================================
Methods:
Public:
performIter:        performs all the iterations inside a for loop
getGradHist:        returns gradient history (default is zero)
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of minibatch SGD
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference:
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.2.1 minibatchSGD

```
import SGD as sgd

eta = 0.025 # learning rate

opt = sgd.minibatchSGD(nSamples = 10,nTotSamples = n,newGrad = 0.0,obj = R,
grad = dR, eta = eta, param = theta, iter = it,
```

```
maxiter=maxIt, objFun=objFun,
gradFun=batchGradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

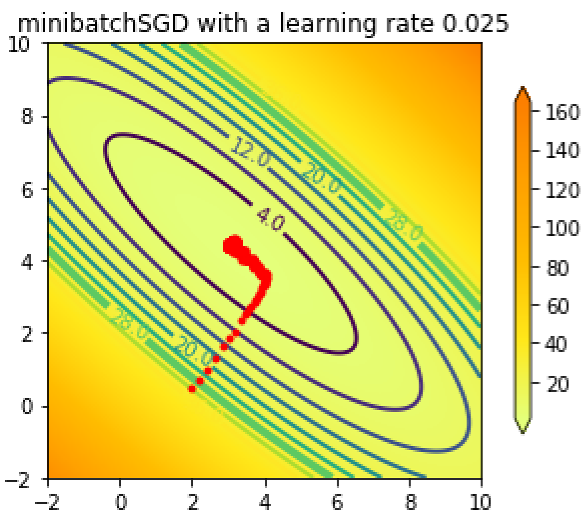
The output is the following:

```
Algorithm:  minibatchSGD

Learning rate =  0.025

Learning schedule:  constant
NOTE: performing a minibatch SGD with  1.0 % of total samples
minibatchSGD |██████████████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 76.85s, Objective = 0.933654
```

If we plot the parameter history:



2.3 RMSprop

```
================================================================================
|                                 RMSprop class                                |
|                derived class from Stochastic Gradient Descent                |
================================================================================
Initialization:
rp = RMSprop(gradHist, updatehist, rho, obj, grad, eta, param,
iter, maxIter, objFun, gradFun, lowerBound, upperBound)
NOTE: gradHist: historical information of gradients
```

```
(array of dimension nparam-by-1)
this should equal to zero for 1st iteration
==============================================================================
Attributes:
grad:           Gradient information (array of dimension nParam-by-1)
eta:            learning rate = 1 by default
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
gradHist:       gradient history accumulator (see the algorithm)
epsilon:        square-root of machine-precision
(required to avoid division by zero)
rho:            exponential decay rate (0.95 may be a good choice)
iter:           iteration number (optional)
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optional input)
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
alg:            algorithm used
__version__:    version of the code
==============================================================================
Methods:
Public:
performIter:performs all the iterations inside a for loop
getGradHist:returns gradient history (default is zero)
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of Adadelta
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Geoffrey  Hinton
```

```
"rmsprop: Divide the gradient by a running average of its recent magnitude."
http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.3.1 RMSprop

```
import SGD as sgd

eta = 0.9 # learning rate

opt = sgd.RMSprop(gradHist=0.0,rho=0.1,obj = R, grad = dR, eta = eta,
param = theta, iter = it, maxiter=maxIt,
objFun=objFun, gradFun=gradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

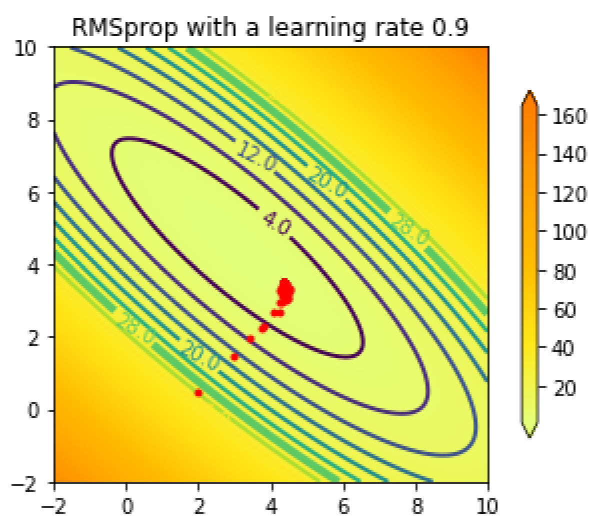
The output is the following:

```
Algorithm:  RMSprop

Learning rate =  0.9

Learning schedule:  constant
RMSprop |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 86.87s, Objective = 1.336377
```

If we plot the parameter history:



2.4 AdaGrad

```
===============================================================================
|                 Adaptive Subgradient Method (AdaGrad) class                  |
|                derived class from Stochastic Gradient Descent                |
===============================================================================
Initialization:
adg = AdaGrad(gradHist, obj, grad, eta, param,
iter, maxIter, objFun, gradFun, lowerBound, upperBound)

NOTE: gradHist:     historical information of gradients
(array of dimension nparam-by-1).
This should equal to zero for 1st iteration
===============================================================================
Attributes:
obj:            Initial objective value (optional input)
grad:           Gradient information (array of dimension nParam-by-1)
eta:            learning rate ( = 1.0, default)
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
gradHist:       sum of gradient history (see the algorithm)
epsilon:        square-root of machine-precision
(required to avoid division by zero)
iter:           iteration number (optional input)
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optional input)
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
alg:            algorithm used
__version__:    version of the code
===============================================================================
Methods:
Public:
performIter:performs all the iterations inside a for loop
getGradHist:returns gradient history (default is zero)
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
```

```
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of AdaGrad
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Duchi, John, Elad Hazan, and Yoram Singer.
"Adaptive subgradient methods for online learning and stochastic optimization."
Journal of Machine Learning Research 12.Jul (2011): 2121-2159.
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.4.1 AdaGrad

```
import SGD as sgd

eta = 0.25 # learning rate

opt = sgd.AdaGrad(gradHist=0.0,obj = R, grad = dR, eta = eta,
param = theta, iter = it, maxiter=maxIt,
objFun=objFun, gradFun=gradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

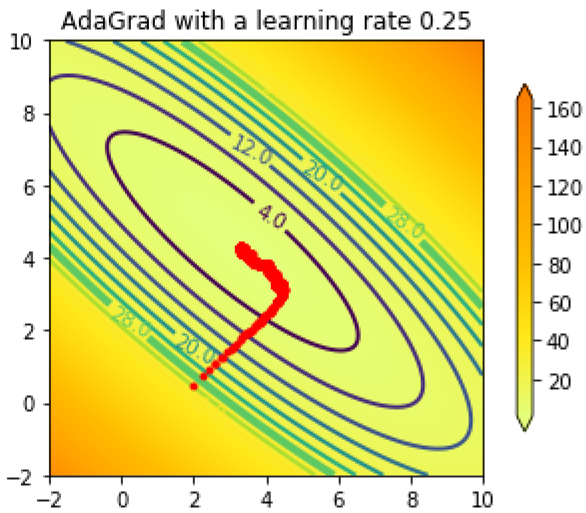
The output is the following:

```
Algorithm:  AdaGrad

Learning rate =  0.25

Learning schedule:  constant
AdaGrad |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 78.16s, Objective = 0.945732
```

If we plot the parameter history:



2.5 Adam

```
==============================================================================
|                    Adaptive moment estimation (Adam) class                 |
|                derived class from Stochastic Gradient Descent              |
==============================================================================
Initialization:
adm = Adam(m, v, beta1, beta2, obj, grad, eta, param,
iter, maxIter, objFun, gradFun, lowerBound, upperBound)

==============================================================================
Attributes:
grad:           Gradient information (array of dimension nParam-by-1)
eta:            learning rate
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
beta1, beta2:   exponential decay rates in [0,1)
(default beta1 = 0.9, beta2 = 0.999)
m:              First moment (array of dimension nParam-by-1)
v:              Second raw moment (array of dimension nParam-by-1)
epsilon:        square-root of machine-precision
(required to avoid division by zero)
iter:           iteration number
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optional input)
upperBound:     upper bound for the parameters (optional input)
```

```
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
alg:            algorithm used
__version__:    version of the code
==============================================================================
Methods:
Public:
performIter:    performs all the iterations inside a for loop
getGradHist:    returns gradient history (default is zero)
getMoments:     returns history of moments
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of Adam
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Kingma, Diederik P., and Jimmy Ba.
"Adam: A method for stochastic optimization."
arXiv preprint arXiv:1412.6980 (2014).
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.5.1 Adam

```
import SGD as sgd

eta = 0.025 # learning rate

opt = Adam(m = 0.0,v = 0.0,beta1 = 0.9,beta2 = 0.999,obj = R, grad = dR,
eta = eta, param = theta, iter = it, maxiter=maxIt,
objFun=objFun, gradFun=gradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

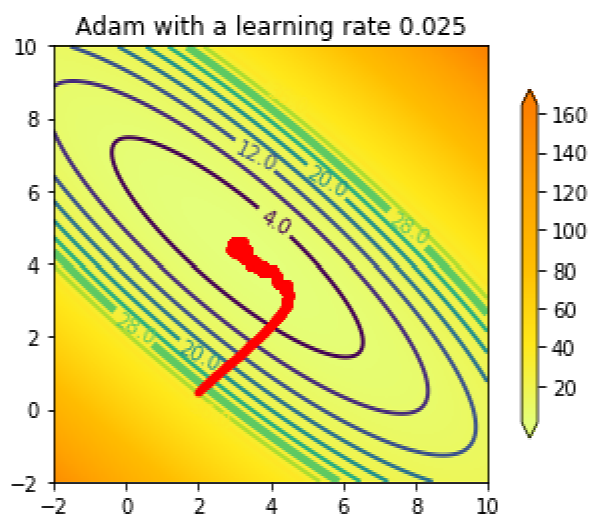
The output is the following:

```
Algorithm:  Adam

Learning rate =  0.025

Learning schedule:  constant
Adam |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 78.51s, Objective = 0.944059
```

If we plot the parameter history:



2.6 Adamax

```
==============================================================================
|                    Adaptive moment estimation (Adamax) class               |
|                 derived class from Stochastic Gradient Descent             |
==============================================================================
Initialization:
admx = Adamax(m, v, beta1, beta2, obj, grad, eta, param,
iter, maxIter, objFun, gradFun, lowerBound, upperBound)

==============================================================================
Attributes: (all private)
grad:           Gradient information (array of dimension nParam-by-1)
eta:            learning rate
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
beta1, beta2:   exponential decay rates in [0,1)
(default beta1 = 0.9, beta2 = 0.999)
```

```
m:              First moment (array of dimension nParam-by-1)
u:              infinity norm constrained second moment
(array of dimension nParam-by-1)
epsilon:        square-root of machine-precision
(required to avoid division by zero)
iter:           iteration number
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optional input)
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
alg:            algorithm used
__version__:    version of the code
==============================================================================
Methods:
Public:
performIter:    performs all the iterations inside a for loop
getGradHist:    returns gradient history (default is zero)
getMoments:     returns history of moments
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of Adam
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Kingma, Diederik P., and Jimmy Ba.
"Adam: A method for stochastic optimization."
arXiv preprint arXiv:1412.6980 (2014).
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.6.1 Adamax

```
import SGD as sgd

eta = 0.25 # learning rate

opt = sgd.Adamax(m = 0.0,u = 0.0,beta1 = 0.9,beta2 = 0.999,obj = R, grad = dR,
eta = eta, param = theta, iter = it, maxiter=maxIt,
objFun=objFun, gradFun=gradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

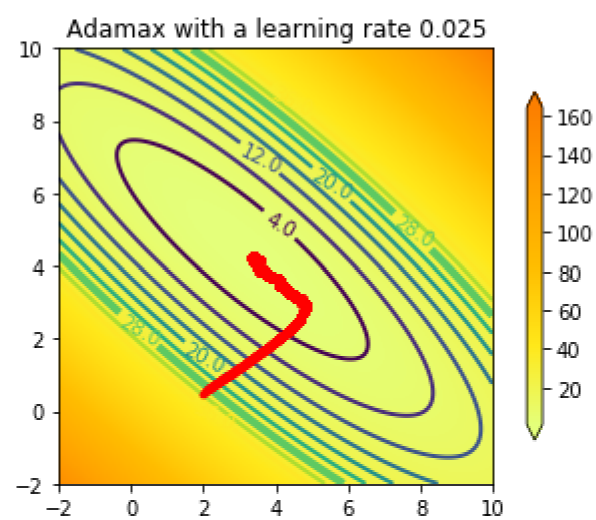
The output is the following:

```
Algorithm:  Adamax

Learning rate =  0.025

Learning schedule:  constant
Adamax |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 79.23s, Objective = 0.952966
```

If we plot the parameter history:



2.7 Nadam

```
================================================================================
|         Nesterov-accelerated Adaptive moment estimation (Nadam) class        |
|               derived class from Stochastic Gradient Descent                 |
```

```
================================================================================
Initialization:
nadm = Nadam(m, v, beta1, beta2, obj, grad, eta, param, iter,
maxIter, objFun, gradFun, lowerBound, upperBound)

================================================================================
Attributes: (all private)
grad:           Gradient information (array of dimension nParam-by-1)
eta:            learning rate
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
beta1, beta2:   exponential decay rates in [0,1)
(default beta1 = 0.9, beta2 = 0.999)
m:              First moment (array of dimension nParam-by-1)
v:              Second raw moment (array of dimension nParam-by-1)
epsilon:        square-root of machine-precision
(required to avoid division by zero)
iter:           iteration number
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optional input)
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
alg:            algorithm used
__version__:    version of the code
================================================================================
Methods:
Public:
performIter:    performs all the iterations inside a for loop
getGradHist:    returns gradient history (default is zero)
getMoments:     returns history of moments
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of Adam
Inherited:
evaluateObjFn:      evaluates the objective function
```

```
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Kingma, Diederik P., and Jimmy Ba.
"Adam: A method for stochastic optimization."
arXiv preprint arXiv:1412.6980 (2014).
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.7.1 Nadam

```
import SGD as sgd

eta = 0.01 # learning rate

opt = sgd.Nadam(m = 0.0,v = 0.0,beta1 = 0.9,beta2 = 0.999,obj = R, grad = dR,
eta = eta, param = theta, iter = it, maxiter=maxIt,
objFun=objFun, gradFun=gradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

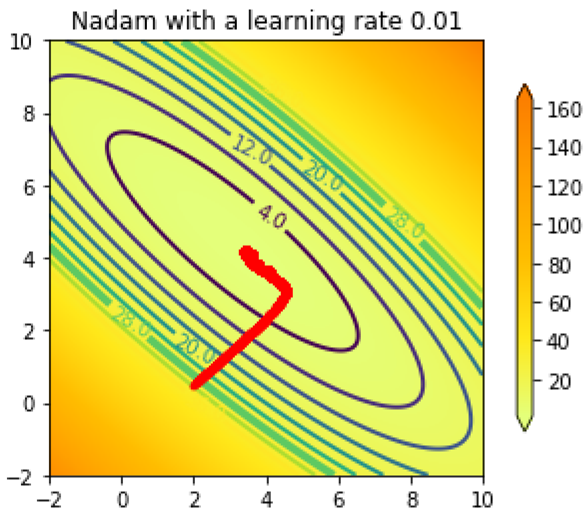
The output is the following:

```
Algorithm:  Nadam

Learning rate =  0.01

Learning schedule:  constant
Nadam |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 78.47s, Objective = 0.961719
```

If we plot the parameter history:



2.8 Adadelta

```
===============================================================================
|                                ADADELTA class                               |
|               derived class from Stochastic Gradient Descent                 |
===============================================================================
Initialization:
add = Adadelta(gradHist, updatehist, rho, obj, grad, eta, param,
iter, maxIter, objFun, gradFun, lowerBound, upperBound)
NOTE: gradHist: historical information of gradients
(array of dimension nparam-by-1)
this should equal to zero for 1st iteration
===============================================================================
Attributes: (all private)
grad:           Gradient information (array of dimension nParam-by-1)
eta:            learning rate = 1 by default
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
gradHist:       gradient history accumulator (see the algorithm)
updateHist:     parameter update history accumulator
epsilon:        square-root of machine-precision
(required to avoid division by zero)
rho:            exponential decay rate (0.95 may be a good choice)
iter:           iteration number (optional)
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optional input)
```

```
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
alg:            algorithm used
__version__:    version of the code
==============================================================================
Methods:
Public:
performIter:performs all the iterations inside a for loop
getGradHist:returns gradient history (default is zero)
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of Adadelta
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Zeiler, Matthew D.
"Adadelta: an adaptive learning rate method."
arXiv preprint arXiv:1212.5701 (2012).
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.8.1 Adadelta

```
import SGD as sgd

eta = 1.0 # learning rate

opt = sgd.Adadelta(gradHist=0.0,updateHist=0.0,rho=0.99,obj = R,
grad = dR, eta = eta, param = theta, iter = it, maxiter=maxIt,
objFun=objFun, gradFun=gradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

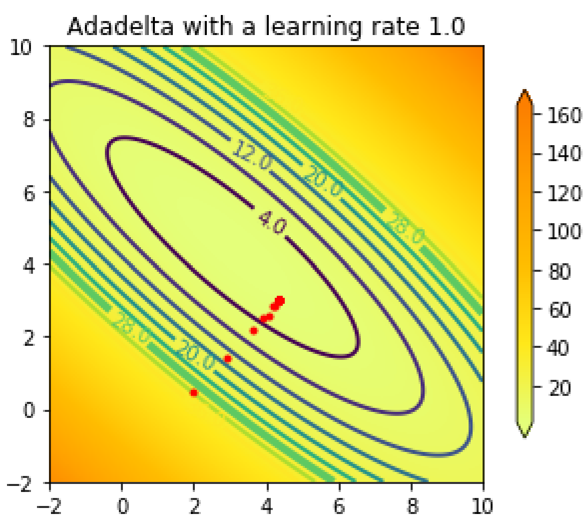
The output is the following:

```
Algorithm:  Adadelta

Learning rate =  1.0

Learning schedule:  constant
Adadelta |████████████████████████████████████| 100.0% Complete:
Time Elapsed = 85.16s, Objective = 1.633017
```

If we plot the parameter history:



2.9 SAG

```
=============================================================================
|                    Stochastic Average Gradient (SAG) class                |
|                derived class from Stochastic Gradient Descent             |
=============================================================================
Initialization:
sag = SAG(nSamples, nTotSamples, fullGrad = 0.0, obj, grad, eta, param,
iter, maxIter, objFun, gradFun, lowerBound, upperBound)

=============================================================================
Attributes: (all private)
fullGrad:           Full gradient information
(array of dimension nParam-by-nTotSamples)
eta:            learning rate
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
nTotSamples:    total number of samples
```

```
nSamples:       number of gradients updated at each iteration
iter:           iteration number (optional)
maxIter:        maximum iteration number (optional input, default = 1)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
lowerBound:     lower bound for the parameters (optional input)
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
learnSched:     learning schedule (constant, exponential or time-based,
default = constant)
lrParam:        learning schedule parameter (default =0.1)
alg:            algorithm used
__version__:    version of the code
==============================================================================
Methods:
Public:
performIter:performs all the iterations inside a for loop
getGradHist:returns gradient history (default is zero)
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:       initialization
update:         performs one iteration of SAG
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Roux, Nicolas L., Mark Schmidt, and Francis R. Bach.
"A stochastic gradient method with an exponential convergence rate
for finite training sets."
Advances in neural information processing systems. 2012.
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.9.1 SAG

```
from StochasticGradientDescent import SAG

eta = 0.0025 # learning rate

opt = SAG(nSamples = 20,nTotSamples= n, obj = R, grad = dR, eta = eta,
param = theta, iter = it, maxiter=maxIt, objFun=objFun,
gradFun=batchGradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

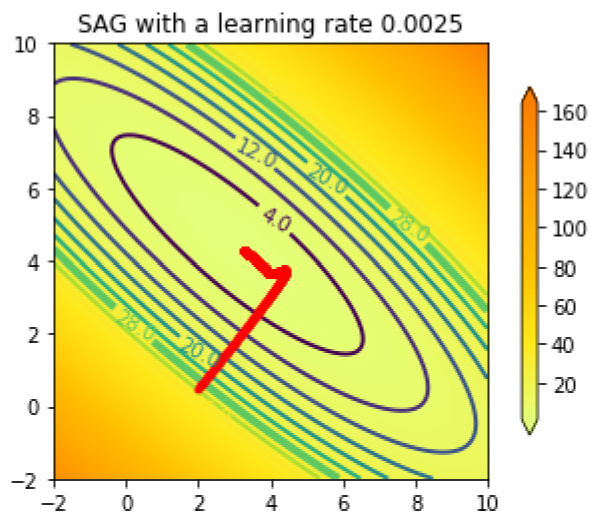
The output is the following:

```
Algorithm:  SAG

Learning rate =  0.0025

Learning schedule:  constant
SAG |██████████████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 77.98s, Objective = 0.940773
```

If we plot the parameter history:



2.10 SVRG

```
===============================================================================
|              Stochastic variance reduced gradient (SVRG) class              |
|               derived class from Stochastic Gradient Descent                |
```

```
================================================================================
Initialization:
opt = SVRG(nTotSamples, innerIter = 10, outerIter = 200, option = 1,obj,
grad, eta, param, iter, maxiter, objFun, gradFun)

NOTE: option = 1 or 2 as suggested in the reference paper.
================================================================================
Attributes:
alg:            SVRG
eta:            learning rate
param:          the parameter vector (array of dimension nParam-by-1)
nParam:         number of parameters
fullGrad:       Full gradient information
(array of dimension nParam-by-nTotSamples)
nTotSamples:    total number of samples
innerIter:      inner iteration
outerIter:      outer iteration
iter:           iteration number (optional input)
maxIter:        maximum iteration number
(optional, default = innerIter*outerIter)
objFun:         function handle to evaluate the objective
(not required for maxit = 1 )
gradFun:        function handle to evaluate the gradient
(not required for maxit = 1 )
mu:             average gradient in the outer iteration
paramBest:      best estimate of the param in the oter iteration
lowerBound:     lower bound for the parameters (optional input)
upperBound:     upper bound for the parameters (optional input)
stopGrad:       stopping criterion based on 2-norm of gradient vector
(default 10^-6)
alg:            algorithm used
__version__:    version of the code
================================================================================
Methods:
Public:
performOuterIter:   performs all the iterations inside a for loop
getGradHist:        returns gradient history (default is zero)
Inherited:
getParam:       returns the parameter values
getObj:         returns the current objective value
getGrad:        returns the current gradient information
getParamHist:   returns parameter update history
Private: (should not be called outside this class file)
__init__:           initialization
innerUpdate:        performs inner iterations of SVRG
```

```
Inherited:
evaluateObjFn:      evaluates the objective function
evaluateGradFn:     evaluates the gradients
satisfyBounds:      satisfies the parameter bounds
learningSchedule:   learning schedule
stopCrit:           check stopping criteria
==============================================================================
Reference: Johnson, Rie, and Tong Zhang.
"Accelerating stochastic gradient descent using predictive variance reduction."
Advances in neural information processing systems. 2013.
==============================================================================
written by Subhayan De (email: Subhayan.De@colorado.edu), July, 2018.
==============================================================================
```

2.10.1 SVRG

```
from StochasticGradientDescent import SVRG

eta = 0.004 # learning rate

opt = SVRG(nTotSamples = n, innerIter = 10, outerIter = 200, option = 1,
obj = R, grad = dR, eta = eta, param = theta, iter = it,
maxiter=maxIt, objFun=objFun, gradFun=batchGradFun) # initialize

opt.performIter() # perform iterations

thetaHist = opt.getParamHist()
```

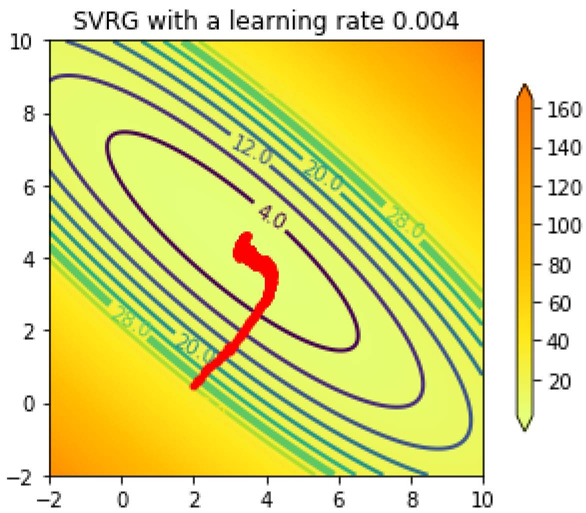
The output is the following:

```
Algorithm:  SVRG

Learning rate =  0.004

Learning schedule:  constant
SVRG |████████████████████████████████████████| 100.0% Complete:
Time Elapsed = 39.94s, Objective = 0.981352
```

If we plot the parameter history:



Report any bugs to Subhayan.De@colorado.edu