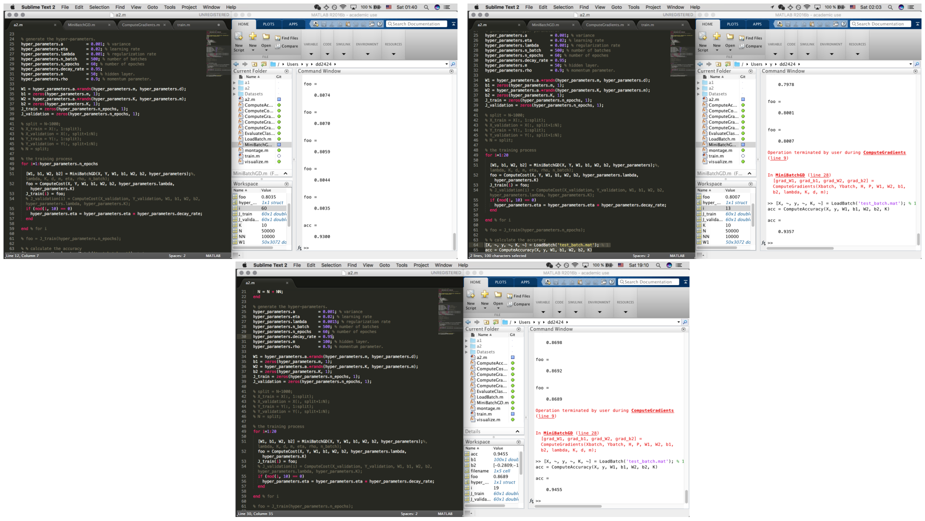
Deep Learning in Data Science
DD2424

Solution to Assignment 2 (for Bonus)

Yiming Fan
yimingf@kth.se



Sublime Text 2 File Edit Selection Find View Goto Tools Project Window Help
a2.m
UNREGISTERED
a2.m MiniBatchGD.m ComputeGradients.m train.m
Line 12, Column 7 Spaces: 2 MATLAB
Sat 01:40
MATLAB R2016b - academic use
HOME PLOTS APPS
Search Documentation
New Script New Open Find Files Compare
FILE VARIABLE CODE SIMULINK ENVIRONMENT RESOURCES
/ Users y dd2424
Current Folder
Command Window
Name Git
a1 a2 Datasets a2.m ComputeAc... ComputeCo... ComputeGra... ComputeGra... ComputeGra... EvaluateClas... LoadBatch.m MiniBatchG... montage.m train.m visualize.m
MiniBatchGD.m (F...
Workspace
Name Value

```
% generate the hyper-parameters.
hyper_parameters.a          = 0.001; % variance
hyper_parameters.eta        = 0.02; % learning rate
hyper_parameters.lambda     = 0.001; % regularization rate
hyper_parameters.n_batch    = 500; % number of batches
hyper_parameters.n_epochs   = 60; % number of epoches
hyper_parameters.decay_rate = 0.95;
hyper_parameters.m          = 50; % hidden layer.
hyper_parameters.rho        = 0.9; % momentum parameter.

W1 = hyper_parameters.a.*randn(hyper_parameters.m, hyper_parameters.d);
b1 = zeros(hyper_parameters.m, 1);
W2 = hyper_parameters.a.*randn(hyper_parameters.K, hyper_parameters.m);
b2 = zeros(hyper_parameters.K, 1);
J_train = zeros(hyper_parameters.n_epochs, 1);
J_validation = zeros(hyper_parameters.n_epochs, 1);

% split = N-1000;
% X_train = X(:, 1:split);
% X_validation = X(:, split+1:N);
% Y_train = Y(:, 1:split);
% Y_validation = Y(:, split+1:N);
% N = split;

% the training process
for i=1:hyper_parameters.n_epochs

  [W1, b1, W2, b2] = MiniBatchGD(X, Y, W1, b1, W2, b2, hyper_parameters);%, lambda, K, d, m, eta, rho, n_batch);
  foo = ComputeCost(X, Y, W1, b1, W2, b2, hyper_parameters.lambda, hyper_parameters.K)
  J_train(i) = foo;
  % J_validation(i) = ComputeCost(X_validation, Y_validation, W1, b1, W2, b2, hyper_parameters.lambda, hyper_parameters.K);
  if (mod(i, 10) == 0)
    hyper_parameters.eta = hyper_parameters.eta * hyper_parameters.decay_rate;
  end

end % for i

% foo = J_train(hyper_parameters.n_epochs);

% % calculate the accuracy
```

```
foo =

    0.8074


foo =

    0.8070


foo =

    0.8059


foo =

    0.8044


foo =

    0.8035


acc =

    0.9300

>>
```

Name	Value
foo	0.8035
hyper_...	1x1 struct
i	60
J_train	60x1 doubl
J_valida...	60x1 doubl
K	10
N	50000
NN	10000
W1	50x3072 do

```
hyper_parameters.a          = 0.001; % variance
hyper_parameters.eta        = 0.02; % learning rate
hyper_parameters.lambda     = 0.001; % regularization rate
hyper_parameters.n_batch    = 500; % number of batches
hyper_parameters.n_epochs   = 60; % number of epoches
hyper_parameters.decay_rate = 0.95;
hyper_parameters.m          = 50; % hidden layer.
hyper_parameters.rho        = 0.9; % momentum parameter.

W1 = hyper_parameters.a.*randn(hyper_parameters.m, hyper_parameters.d);
b1 = zeros(hyper_parameters.m, 1);
W2 = hyper_parameters.a.*randn(hyper_parameters.K, hyper_parameters.m);
b2 = zeros(hyper_parameters.K, 1);
J_train = zeros(hyper_parameters.n_epochs, 1);
J_validation = zeros(hyper_parameters.n_epochs, 1);

% split = N-1000;
% X_train = X(:, 1:split);
% X_validation = X(:, split+1:N);
% Y_train = Y(:, 1:split);
% Y_validation = Y(:, split+1:N);
% N = split;

% the training process
for i=1:20

  [W1, b1, W2, b2] = MiniBatchGD(X, Y, W1, b1, W2, b2, hyper_parameters);%, lambda, K, d, m, eta, rho, n_batch);
  foo = ComputeCost(X, Y, W1, b1, W2, b2, hyper_parameters.lambda, hyper_parameters.K)
  J_train(i) = foo;
  % J_validation(i) = ComputeCost(X_validation, Y_validation, W1, b1, W2, b2, hyper_parameters.lambda, hyper_parameters.K);
  if (mod(i, 10) == 0)
    hyper_parameters.eta = hyper_parameters.eta * hyper_parameters.decay_rate;
  end

end % for i

% foo = J_train(hyper_parameters.n_epochs);

% % calculate the accuracy
[X, ~, y, ~, K, ~] = LoadBatch('test_batch.mat'); % 1
acc = ComputeAccuracy(X, y, W1, b1, W2, b2, K)
```

```
    0.7978


foo =

    0.8001


foo =

    0.8007

Operation terminated by user during ComputeGradients
(line 9)


In MiniBatchGD (line 28)
  [grad_W1, grad_b1, grad_W2, grad_b2] =
  ComputeGradients(Xbatch, Ybatch, H, P, W1, W2, b1,
  b2, lambda, K, d, m);

>> [X, ~, y, ~, K, ~] = LoadBatch('test_batch.mat'); % 1
acc = ComputeAccuracy(X, y, W1, b1, W2, b2, K)

acc =

    0.9357

>>
```

Name	Value
foo	0.8007
hyper_...	1x1 struct
i	13
J_train	60x1 doubl
J_valida...	60x1 doubl
K	10
N	50000
NN	10000
W1	50x3072 do

```
  N = N + NN;
end

% generate the hyper-parameters.
hyper_parameters.a          = 0.001; % variance
hyper_parameters.eta        = 0.02; % learning rate
hyper_parameters.lambda     = 0.0015; % regularization rate
hyper_parameters.n_batch    = 500; % number of batches
hyper_parameters.n_epochs   = 60; % number of epoches
hyper_parameters.decay_rate = 0.95;
hyper_parameters.m          = 100; % hidden layer.
hyper_parameters.rho        = 0.9; % momentum parameter.

W1 = hyper_parameters.a.*randn(hyper_parameters.m, hyper_parameters.d);
b1 = zeros(hyper_parameters.m, 1);
W2 = hyper_parameters.a.*randn(hyper_parameters.K, hyper_parameters.m);
b2 = zeros(hyper_parameters.K, 1);
J_train = zeros(hyper_parameters.n_epochs, 1);
J_validation = zeros(hyper_parameters.n_epochs, 1);

% split = N-1000;
% X_train = X(:, 1:split);
% X_validation = X(:, split+1:N);
% Y_train = Y(:, 1:split);
% Y_validation = Y(:, split+1:N);
% N = split;

% the training process
for i=1:20

  [W1, b1, W2, b2] = MiniBatchGD(X, Y, W1, b1, W2, b2, hyper_parameters);%, lambda, K, d, m, eta, rho, n_batch);
  foo = ComputeCost(X, Y, W1, b1, W2, b2, hyper_parameters.lambda, hyper_parameters.K)
  J_train(i) = foo;
  % J_validation(i) = ComputeCost(X_validation, Y_validation, W1, b1, W2, b2, hyper_parameters.lambda, hyper_parameters.K);
  if (mod(i, 10) == 0)
    hyper_parameters.eta = hyper_parameters.eta * hyper_parameters.decay_rate;
  end

end % for i

% foo = J_train(hyper_parameters.n_epochs);
```

```
    0.8698


foo =

    0.8692


foo =

    0.8689

Operation terminated by user during ComputeGradients
(line 9)


In MiniBatchGD (line 28)
  [grad_W1, grad_b1, grad_W2, grad_b2] =
  ComputeGradients(Xbatch, Ybatch, H, P, W1, W2, b1,
  b2, lambda, K, d, m);

>> [X, ~, y, ~, K, ~] = LoadBatch('test_batch.mat'); % 1
acc = ComputeAccuracy(X, y, W1, b1, W2, b2, K)

acc =

    0.9455

>>
```

Name	Value
acc	0.9455
b1	100x1 doub
b2	[-0.2809;-1
filename	1x5 cell
foo	0.8689
hyper_...	1x1 struct
i	19
J_train	60x1 doubl
J_valida...	60x1 doubl

1 Optimize the performance of the network

We tried the following methods of optimization:

1. Trained the network with more than 60 epochs until the network converges (i.e. when the gradient stops decreasing). We used the last 1000 data for validation.

2. We did a finer grid search and found the 'better' combination of hyper-parameters: $\lambda = 0.001$, $\eta = 0.02$. The learning rate η decays by 0.95 after every 10 epochs. This set of hyper-parameters achieved more than **93%** and ranked 1st on the *leaderboard* from 21st April.

3. We tried increasing the number of hidden nodes from 50 to 100 and also increase the regularization term λ respectively. When the $\#(hidden\,nodes) = 100$ the network converged after 59 epochs and achieved **94.55%** accuracy. **Conclusion:** Increasing the number of hidden nodes will increase the complexity of the network, hence improve the accuracy. However too many hidden nodes may cause the network to overfit.

Some screenshots of the optimization are shown.

2 Train network using a different activation to ReLu

We tried `sigmoid` function for activation function. The training result was terrible:

1. With momentum the cross-entropy loss doesn't decrease - the network is worse than random guesser.

2. Without momentum the loss converge to around 2.2 after 5 epochs - a little better than random guesser.

3. `sigmoid` is usually 2~3 times slower than `ReLu`.

Conclusion: `sigmoid` is slow and inefficient. We will discard using it in the future.