

COMP 790-125, HW4

Wile E. Coyote

November 25, 2014

Deadline: 12/6/2014 11:59PM EST

Submit hw4.pdf by e-mail, <mailto:vjojic+comp790+hw4@cs.unc.edu>.

Instructions

Software In most cases the department and university machines come with software that we need pre-installed. If not you will need to install

1. a \LaTeX implementation, on Windows MikTeX <http://miktex.org/>, on Mac MacTeX <http://www.tug.org/mactex/2011/>
2. MATLAB following instructions from <http://software.unc.edu>
3. RECOMMENDED: git to keep version of your homework files as you work on them <http://git-scm.com/downloads/>

If you are working on a department/university machine some or all of this may already be installed.

Files for HW4 Download <http://www.cs.unc.edu/~vjojic/comp790/hw4.zip> and unpack it into a directory. In the directory you unpacked the zip file, you will find several .m,.tex,.sty files and hw4.pdf. You will be changing hw4.tex and adding some pdf generated by MATLAB scripts into this directory. You should learn MATLAB and \LaTeX , but for this assignment instructions will be pretty straightforward. If you find yourself stuck on how to write out a particular piece of math in \LaTeX look at <http://tobi.oetiker.ch/lshort/lshort.pdf>.

Problems and points Maximum number of points that contribute to your grade from this homework is 10. If you earn more than 10 points, this will be recorded, but it will not affect your grade.

Read all the problems CAREFULLY, some will have an obvious answer box which means that you will find `\answer` in hw4.tex and replace it with your answer. Others will have you change something in a piece of code or may require you to run a piece of code and include a .pdf in the hw4.tex. Hence, not all the problems will have explicit and obvious answer box. As a rule of

thumb EVERY problem will have you change the hw4.tex. If you completed a problem, but have not changed hw4.tex, go back and read the problem you are forgetting something. Once you have changed hw4.tex remake hw4.pdf and look at it. Make sure that your answer has been updated and it looks clean and readable. Keep copies of hw4.tex so you can easily revert to an older version (subversion is fine for this).

Turning the homework in You will send me hw4.pdf file by e-mail. If you send me multiple versions of the document I will read the latest version that arrived prior to the deadline.

1 Getting the data

The homework archive has a separate subdirectory with Matlab scripts you need to get started, called `work`. From <http://yann.lecun.com/exdb/mnist/> download:

- `train-images-idx3-ubyte.gz`
- `train-labels-idx1-ubyte.gz`

decompress these files and add them to your work directory. In Matlab change to the work directory and do following

```
mnistData = loadMNISTImages('train-images.idx3-ubyte');
mnistLabels = loadMNISTLabels('train-labels.idx1-ubyte');
imagesc(reshape(mnistData(:,1),[28 28]))
colormap(gray)
mnistLabels(1)
```

You should see an image of handwritten digit 5 and this should match printed value for `mnistLabels(1)`.

2 Neural network architecture

We will develop a method for digit recognition. The data will consist of examples of handwritten digits of size 28×28 gray pixels, each such image labeled with digit that it corresponds to. We will use a neural network consisting of four layers.

The top layer is composed of indicator variables, one for every digit. Only one of those variables is 1, the rest will be zero. So, if $h_{1,i} = 1$ then the digit is $i - 1$.

The bottom layer is composed of pixel intensity variables. Since images are of size 28 by 28 , the total number of variables in that layer is 784 .

We use n_l to denote number of nodes in layer l where they are not known ahead of time. In Fig. 1 one of the edges is labeled by its weight, $w_{n_2,784}^2$. We

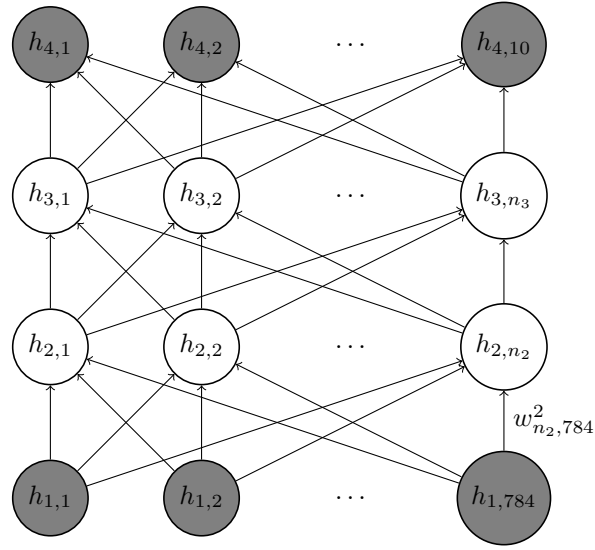


Figure 1: Network is composed of 4 layers. Bottom layer corresponds to a handwritten digit image, one node per pixel of the 28 by 28 image. Top layer is composed of 10 indicator variables, one for each possible digit. Two neighboring layers are densely connected. We label a single edge to illustrate notation for weights. Additional details are provided in text.

use $w_{i,j}^l$ to denote a weight of input from node j in layer $l - 1$ to node i in layer l .

Each layer is densely connected to neighboring layers.

3 Notation

We will adopt notation whereby layer index is denoted in the superscript and subscript indices refer to nodes

- $w_{i,j}^l$ weight of connection between node j in layer $l - 1$ and node i in layer l
- b_i^l is bias of node i in layer l
- f^L is softmax, f^{L-1}, \dots, f^2 are unit activation functions
- input to node i in layer l

$$a_i^l = \sum_j w_{i,j}^l o_j^{l-1} + b_i^l \quad (1)$$

- output from node i in layer l

$$o_i^l = f^l(a_i^l) \quad (2)$$

Hence o_j^1 is intensity of j^{th} pixel in the image of a handwritten digit, and o_j^L is an indicator of whether the handwritten digit is $j - 1$. Note that the first layer does not have corresponding \mathbf{w} or b values because it is not modeled – we assume that image pixel intensities are known. We will use n_l to denote number of nodes in layer l .

We will reserve index t to indicate sample index. Note that activations and outputs may differ for different samples. Where relevant we will use $a_i^{l,t}, o_i^{l,t}$ to denote sample specific analogs of the input and output variables defined above. Finally, x^T will denote transposed vector, and $f'(x)$ will denote first derivative of a univariate function.

Problem 1(1pt) Using n_l and definitions of different variables in the notation Section 3 give their counts

- \mathbf{b}^l is a vector of size $n_l \times 1$
- \mathbf{W}^l is a matrix of size answer
- \mathbf{a}^l is a vector of size answer
- \mathbf{o}^l is a vector of size answer

4 Softmax activation function

Softmax activation function is given by

$$f_k(\mathbf{a}) = \frac{\exp\{a_k\}}{\sum_{c=1}^C \exp\{a_c\}} \quad (3)$$

where $k \in \{1, \dots, C\}$.

Using (1) we can write

$$o_k^l = \frac{\exp\{(\mathbf{w}_k^L)^T \mathbf{o}^{l-1}\}}{\sum_{c=1}^C \exp\{(\mathbf{w}_c^L)^T \mathbf{o}^{l-1}\}} \quad (4)$$

Note the sum of exponential functions, this should always signal use of log-sum.

5 Rectified linear activation function

A rectified linear unit activation function has following form:

$$f(a) = \max(0, a)$$

Problem 2(1pt) Plot this function on a interval $[-1, 1]$ and replace `emptiness.pdf` with appropriate pdf below.

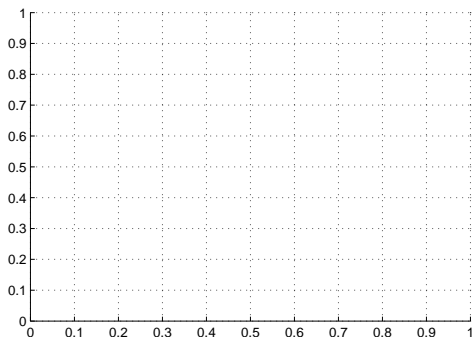


Figure 2: This is emptiness, it earns no points.

6 Forward Propagation

In our network forward propagation starts from the layer with pixel intensities and propagates activations through the rectified linear units (ReLUs), and

finally through softmax. Specifically, activation functions for each layer are

$$\begin{aligned} f^2(a) = f^3(a) &= \max(0, a) \\ f_k^4(\mathbf{a}) &= \frac{\exp\{a_k\}}{\sum_{c=1}^C \exp\{a_c\}} \end{aligned}$$

ReLU units in a single layer produce outputs independently of each other. However, softmax units produce outputs jointly and depend on all the inputs into their layer, whole vector \mathbf{a} . This is due to the denominator depending on all a_c s. It stands to reason, since the softmax units will sum up to 1, and hence are not be independent.

Problem 3(1pt) Implement a function that takes performs forward propagation in the above network. It should return outputs of each layer of neural net. You only need to implement propagation for ReLU units (layers 2 through L-1).

```
function os = forwardProp(data, layerwiseParams, architecture)
L = length(architecture);

os = cell(L,1);
os{1} = data;
N = size(data,2);
% relu propagation
for l=2:L-1
    W = layerwiseParams(l).w;
    b = layerwiseParams(l).b;
    a = W*os{l-1} + repmat(b,[1 N]);
    os{l} = ... % implement activation function of for
                % rectified linear unit
end

% softmax
softmaxW = layerwiseParams(L).w;
C = architecture(L).size; % number of classes
o = softmaxW * os{L-1};
o = o - repmat(max(o,[],1),[C 1]);
o = exp(o);
o = o./repmat(sum(o),[C 1]);
os{L} = o;
```

In the case of digits, data is a matrix of size $784 \times N$. Note that propagation is performed for all samples simultaneously using matrix multiplication – there is no for loop over samples.

Look at script `wrk.m` for examples of input provided to this code.

7 Loss function

Next we will formulate a loss function that will tell us how good is our neural network in predicting the digits.

Problem 4(1pt) Loss function for softmax model for a single sample is given by

$$\begin{aligned} E_t(\mathbf{W}) &= \sum_{c=1}^C [y^t = c] \log o_c^{L,t} \\ &= \sum_{c=1}^C [y^t = c] \left((\mathbf{w}_c^L)^T \mathbf{o}^{L-1,t} - \log \sum_{k=1}^C \exp \{ (\mathbf{w}_k^L)^T \mathbf{o}^{L-1,t} \} \right) \end{aligned}$$

where

$$[v] = \begin{cases} 1, & \text{if } v \text{ is true} \\ 0, & \text{otherwise.} \end{cases}$$

Write out gradient of the loss function with respect to weight $w_{c,i}^L$

$$\frac{\partial E_t(\mathbf{W})}{\partial w_{s,i}^L} = \boxed{\text{answer}}$$

Hint: Use (4) to simplify, the fact that $[\text{not } v] = 1 - [v]$. The solution is very simple.

Write out gradient

$$\nabla_{\mathbf{w}_c^L} E_t(\mathbf{W}) = \boxed{\text{answer}}$$

Average loss function for given a dataset composed of N samples (y, \mathbf{x}) is given by

$$E(\mathbf{W}) = \frac{1}{N} \sum_{t=1}^N \sum_{c=1}^C [y^t = c] \log o_c^{L,t}$$

Write out gradient of the loss function with respect to weights \mathbf{w}_c^L

$$\nabla_{\mathbf{w}_c^L} E(\mathbf{W}) = \boxed{\text{answer}}$$

Problem 5(1pt) Add ridge penalty $\frac{\lambda}{2} \sum_i \sum_c w_{c,i}^2$ to the average loss function. Sometimes this penalty is also referred to as *weight decay*. Penalized average log-likelihood is then

$$E(\mathbf{W}, \lambda) = \frac{1}{N} \sum_{t=1}^N \sum_{c=1}^C [y^t = c] \log o_c^{L,t} + \lambda \sum_{c=1}^C \sum_{i=1}^p (w_{c,i}^L)^2$$

What is the difference between penalizing average loss vs penalizing sum of losses – meaning with and without division by N ? Why would it be beneficial to do one or the other? answer

Problem 6(1pt) Change your gradient to reflect the added penalty. It is not a large change.

$$\nabla_{\mathbf{w}_c^L} E(\mathbf{W}, \lambda) = \boxed{\text{answer}}$$

Conclude how parameter gradient changes if weight decay on parameters is introduced. The gradient $\boxed{\text{answer}}$.

Problem 7(1pt) Here is a function that compute average loss for soft-max model. Add code for computing gradient

```
function [pall,grad] = softmaxLoss(y,Wvec,X,lambda,C)
N = length(y);
labelMatrix = full(sparse(y,1:length(y),1)); % class indicator matrix
W = reshape(Wvec,[C length(Wvec(:))/C]);
p = W*X;
p = p - repmat(max(p,[],1),[C 1]);
p = exp(p);
p = p./repmat(sum(p),[C 1]);
pall = 1/N*sum(sum(labelMatrix.*log(p))) + lambda/2*sum(sum(W(:).^2));
grad = ...
grad = grad(:);
```

Make sure that your gradient is organized in the same way that matrices are unrolled in Matlab. To get a better idea do this

```
W = [1 2 3; 4 5 6]
W(:)
```

Note that the unrolled W is composed of columns of matrix W.

Problem 8(1pt) Use finite differences trick to check your gradient. Note that this code assumes that you ordered partial derivatives in your gradient correctly.

```
function checkGradient(f,x)
[~, grad] = f(x);
d = zeros(length(x),1);
epsilon = 1e-4;
fd = zeros(length(x),1);
for i=1:length(x)
    d(i) = epsilon;
    fd(i) = (f(x+d) - f(x-d))/(2*epsilon);
    d(i) = 0;
end
norm(grad - fd)/norm(grad + fd)
fprintf('this value should be small\n');
```

Run this to confirm that your gradient is accurate.


```

N = 10; p = 5; C = 3;
y = randi(C,1,N);
X = randn(p,N);
W0 = randn(C,p);
lambda = 0;
checkGradient(@(W) softmaxLoss(y,W(:),X,lambda,C), W0(:))
lambda = 0.2;
checkGradient(@(W) softmaxLoss(y,W(:),X,lambda,C), W0(:))

```

If the first test does not work, your softmax gradient is wrong. If *only* the second test does not work then your ridge penalty gradient is wrong. Give output of the code above. answer

Problem 9(1pt) Subdifferential is defined as

$$\partial f(x_0) = \{g | f(x) - f(x_0) \geq g(x - x_0)\}$$

Geometrically, these are slopes of lines that pass through $(x_0, f(x_0))$ and are always below the function. It might help to look at the Fig. 2 to figure this out geometrically.

Give subdifferential for the rectified linear unit activation function

$$\partial_a \max(0, a) = \begin{cases} \boxed{\text{answer}}, & \boxed{\text{answer}} \\ \boxed{\text{answer}}, & \boxed{\text{answer}} \\ \boxed{\text{answer}}, & \boxed{\text{answer}}. \end{cases}$$

8 Backprop derivation

The loss for a network on a single sample is

$$E_t(\mathbf{W}) = \sum_c [y^t = c] \log o_c^L = \sum_c [y^t = c] \log f_c^L(\mathbf{a}^L) \quad (5)$$

Here we took into account the fact that each softmax unit output depends on input into all softmax units by providing argument \mathbf{a} to f^L .

Problem 10(1pt) Express a_j^{l+1} in terms of $\mathbf{a}^l, \mathbf{w}^{l+1}, b_j^l$ and obtain partial derivative with respect to a_i^l . You can leave derivative of f^l as just $(f^l)'$

$$\begin{aligned} a_j^{l+1} &= \boxed{\text{answer}} \\ \frac{\partial a_j^{l+1}}{\partial a_i^l} &= \boxed{\text{answer}} \end{aligned}$$

Average loss on the whole training set is given by

$$E(\mathbf{W}) = \frac{1}{N} \sum_{t=1}^N E_t(\mathbf{W})$$

and penalized average loss is given by

$$E(\mathbf{W}, \lambda) = E(\mathbf{W}) + \frac{\lambda}{2} \sum_{l=2}^L \sum_{j=1}^{n_l} (w_{i,j}^l)^2.$$

We wish to obtain partial derivatives with respect to $w_{i,j}^l$ and b_i^l

$$\frac{\partial E_t}{\partial w_{i,j}^l} = \frac{\partial E_t}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_{i,j}^l} \quad (6)$$

$$\frac{\partial E_t}{\partial b_i^l} = \frac{\partial E_t}{\partial a_i^l} \frac{\partial a_i^l}{\partial b_i^l}. \quad (7)$$

Dependence of error on unit's activation will be deemed δ_i^l

$$\delta_i^l = \frac{\partial E_t}{\partial a_i^l} \quad (8)$$

Problem 11(1pt) Size of vector δ^l is

Problem 12(1pt) Apply (1) and (8), in (6) and in (7) to rewrite partial derivatives in terms of δ 's and a^{l-1} s

$$\frac{\partial E_t}{\partial w_{i,j}^l} = \text{answer}$$

$$\frac{\partial E_t}{\partial b_i^l} = \text{answer}$$

Problem 13(1pt) Use (3),(8),(5) and the fact that f^L is softmax to obtain

$$\delta_c^L = \frac{\partial E_t}{\partial a_{L,c}} = \text{answer}$$

Problem 14(1pt) Assume $l < L$. In a forward propagation, output of node i in layer l affects multiple nodes in layer $l+1$. Express δ_i^l in terms of partial derivatives $\frac{\partial E_t}{\partial a_j^{l+1}}$ and $\frac{\partial a_j^{l+1}}{\partial a_i^l}$

$$\delta_i^l = \text{answer}$$

Now use definition of δ_j^{l+1} , and previously derived $\frac{\partial a_j^{l+1}}{\partial a_i^l}$ to express δ_i^l

$$\delta_i^l = \text{answer}$$

Problem 15(1pt) Still assuming $l < L$, use the fact that each f^l is a rectified linear unit and its subdifferential you obtained earlier to express δ_i^l

$$\delta_i^l = \boxed{\text{answer}}$$

You can make a simplifying assumption and use just a single subgradient at a point where subdifferential is a set with more than one real value.

Problem 16(1pt) Implement a function that computes gradient of the penalized loss with respect to w s and b using backprop. Use `layerwiseParams(l).w` and `layerwiseParams(l).b` to access parameters for l^{th} layer. Note that \mathbf{o}^l values are stored in `os{l}`. You will need those.

```
function [ cost, grad ] = deepReluSoftMaxCost(params, architecture,...
                                             data,labels,lambda)

L = length(architecture);
[layerwiseParams] = unpackParams(params,architecture);
paramStruct = struct('w',[],'b',[]);
layerwiseParamsGrad = repmat(paramStruct,[L 1]);
for l = 2:L
    layerwiseParamsGrad(l).w = zeros(size(layerwiseParams(l).w));
    layerwiseParamsGrad(l).b = zeros(size(layerwiseParams(l).b));
end
cost = 0;
T = size(data, 2);
groundTruth = full(sparse(labels, 1:T, 1));

% forward propagation
os = forwardProp(layerwiseParams,architecture,data);
cost = cost -1/T*groundTruth(:)'*log(os{L}(:)) + lambda/2*sum(layerwiseParams(L).w(:).^2);

delta{L} = ...
assert(size(delta{L},1) == architecture(L).size);

for l=L-1:-1:2
    delta{l} = ...
    assert(size(delta{l},1) == architecture(l).size);
    cost = cost + lambda/2*sum(layerwiseParams(l).w(:).^2);
end

for l=L:-1:2
    layerwiseParamsGrad(l).w = ...
    if size(layerwiseParamsGrad(l).b)
        layerwiseParamsGrad(l).b = ...
    end
end
end
```

```

for l=1:L
    assert(all(size(layerwiseParamsGrad(l).w) == size(layerwiseParams(l).w)))
    assert(all(size(layerwiseParamsGrad(l).b) == size(layerwiseParams(l).b)))
end
grad = packParams(layerwiseParamsGrad);

```

Once completed run

```

d = 3;
nametag = date;
trainData = normalizeData(randn(d^2,21));
trainLabels = [ones(1,4) 2*ones(1,4) 3*ones(1,4) 4*ones(1,4) 5*ones(1,5)];

arch(1) = struct('size',d^2,'hasb',0,'hasw',0);
%first relu layer
arch(2) = struct('size',3,'hasb',1,'hasw',1);
%second relu layer
arch(3) = struct('size',4,'hasb',1,'hasw',1);
%softmax layer -- no bias, 10 indicator variables one for each digit
arch(4) = struct('size',5,'hasb',0,'hasw',1);
lambda = 0.001;
params = initLayerwiseParams(arch);
f = @(p) deepReluSoftMaxCost(p,arch,trainData, trainLabels,lambda);
checkGradient(f,packParams(params))

```

Value reported by the call to `checkGradient` should be smaller than 10^{-8} .

Problem 17(1pt) If the gradient check passes, run

```

addpath ./minFunc/

mnistData = loadMNISTImages('train-images.idx3-ubyte');
mnistLabels = loadMNISTLabels('train-labels.idx1-ubyte');

labeledSet = find(mnistLabels >= 0 & mnistLabels <= 9);

numTrain = round(numel(labeledSet)/2);
trainSet = labeledSet(1:numTrain);
testSet = labeledSet(numTrain+1:end);

trainData = normalizeData(mnistData(:, trainSet));
trainLabels = mnistLabels(trainSet)' +1;

testData = normalizeData(mnistData(:, testSet));
testLabels = mnistLabels(testSet)' +1;

```

```

clear arch
%input layer, digit image 28*28, no params
arch(1) = struct('size',28*28,'hasb',0,'hasw',0);
%first relu layer
arch(2) = struct('size',64,'hasb',1,'hasw',1);
%second relu layer
arch(3) = struct('size',64,'hasb',1,'hasw',1);
%softmax layer -- no bias, 10 indicator variables one for each digit
arch(4) = struct('size',10,'hasb',0,'hasw',1);

L = length(arch);
params0 = initLayerwiseParams(arch);
optParams = trainDeepReluSoftmax(arch,trainData,trainLabels,params0,0.0001)

os0 = forwardProp(params0,testData);
[~,pred0] = max(os0{L},[],1);
err0 = sum(pred0 ~= testLabels)/length(testLabels);

os1 = forwardProp(optParams,testData);
[~,pred1] = max(os1{L},[],1);
err1 = sum(pred1 ~= testLabels)/length(testLabels);

fprintf('Error before training: %d\n Error after training: %d\n', err0,err1);

Report errors before and after training answer .

```