COMPSCI 527 Homework 4

Cody Lieu, Yixin Lin

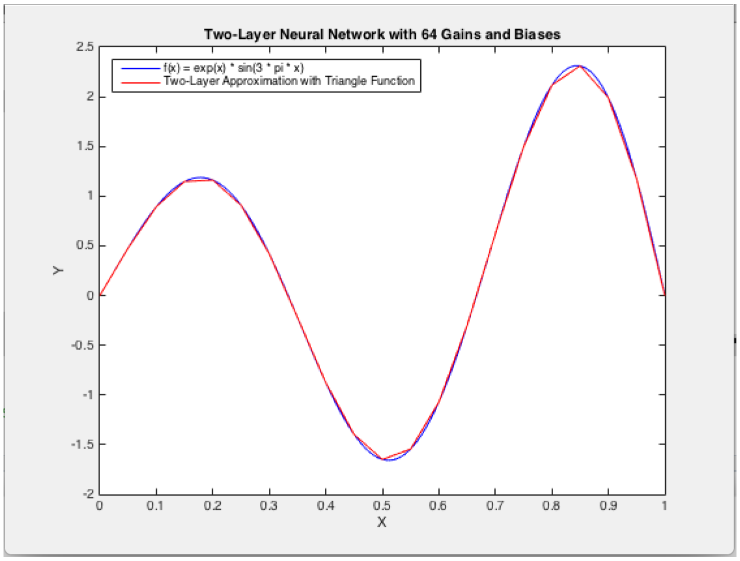
October 29, 2015

Problem 1(a)

```
function X = nn(X, net)

for i = 1:size(net, 2)
    curNet = net(i);
    W = [curNet.gain curNet.bias];
    X = [X; ones(1, size(X, 2))];
    a = W*X;
    X = curNet.h(a);
end

end
```

Problem 1(b)

```
function net = approximator(f, T)

k = floor(1/T) + 1;
sample = linspace(0, 1, k);
newT = sample(2) - sample(1);

ReLU = @(x) max(0, x);
triangle = @(x) ReLU(x + 1) - 2 .* ReLU(x) + ReLU(x - 1);

reluNet.gain = zeros(k, 1) + 1 ./ newT;
reluNet.bias = transpose(-(0:k-1));
reluNet.h = triangle;

identityNet.gain = f(sample);
identityNet.bias = 0;
identityNet.h = @(x) x;

net(1) = reluNet;
net(2) = identityNet;

end
```



Problem 1(c)

$n_w = 3K + 1$

Problem 2(a)

$p = m - n + 1$

Problem 2(b)

$$C_v(\mathbf{k}) = \begin{bmatrix} k_3 & k_2 & k_1 & & & & \\ & k_3 & k_2 & k_1 & & & \\ & & k_3 & k_2 & k_1 & & \\ & & & k_3 & k_2 & k_1 & \\ & & & & k_3 & k_2 & k_1 \end{bmatrix}$$

$$C_f(\rho(\mathbf{k})) = \begin{bmatrix} k_3 & & & & \\ k_2 & k_3 & & & \\ k_1 & k_2 & k_3 & & \\ & k_1 & k_2 & k_3 & \\ & & k_1 & k_2 & k_3 \\ & & & k_1 & k_2 \\ & & & & k_1 \end{bmatrix}$$

$a(\mathbf{x}) = \mathbf{x} *_v \mathbf{k} + b$

$a(\mathbf{x}) = C_v(\mathbf{k})\mathbf{x} + b$

Differentiate both sides:

$A_{\mathbf{x}} = C_v(\mathbf{k})$

$A_{\mathbf{x}}^T = C_v(\mathbf{k})^T$

$A_{\mathbf{x}}^T = C_f(\rho(\mathbf{k}))$

$e_{\mathbf{x}} = A_{\mathbf{x}}^T e_{\mathbf{a}}$

$e_{\mathbf{x}} = C_f(\rho(\mathbf{k}))e_{\mathbf{a}}$

$e_{\mathbf{x}} = e_{\mathbf{a}} *_f \rho(\mathbf{k})$

Problem 2(c)

$\mathbf{a}' = \sigma_r(\mathbf{a}(\mathbf{x}), 2) = [a_1, a_3, a_5]^T$

$e'_{\mathbf{a}} = \sigma_r(\mathbf{a}(\mathbf{x}), 2) = [e_{a1}, e_{a3}, e_{a5}]^T$

$e'_{\mathbf{x}} = \delta(e'_{\mathbf{a}}, p) *_f \rho(\mathbf{k})$

$e'_{\mathbf{x}} = C_f(\rho(\mathbf{k}))\delta(e'_{\mathbf{a}}, p)$

$e'_{\mathbf{x}} = C_v(\mathbf{k})^T\delta(e'_{\mathbf{a}}, p)$

$e'_{\mathbf{x}} = A_{\mathbf{x}}^T\delta(e'_{\mathbf{a}}, p)$

The original equivalence of $e_{\mathbf{x}} = A_{\mathbf{x}}^T e_{\mathbf{a}}$ becomes $e'_{\mathbf{x}} = \sigma_r(A_{\mathbf{x}}, 2)^T e'_{\mathbf{a}}$, which makes sense since every 2nd row of $A_{\mathbf{x}}$ is removed in the matrix multiplication with the transpose of $A_{\mathbf{x}}$ since every 2nd row of $e_{\mathbf{a}}$ was removed with the row sampling operator. These two expressions result in the equivalence below:

$A_{\mathbf{x}}^T\delta(e'_{\mathbf{a}}, p) = \sigma(A_{\mathbf{x}}, 2)^T e'_{\mathbf{a}}$

This equivalence makes intuitive sense. On the left-hand side of the expression, we have the transpose of the Jacobian multiplied by $e_{\mathbf{a}}$ after being row-sampled with a stride of 2 and then diluted to p. This means that the left-hand side of the expression is equivalent to $A_{\mathbf{x}}^T e_{\mathbf{a}}$ from part b except every 2nd column of $A_{\mathbf{x}}^T$ (2nd row of $A_{\mathbf{x}}$) contributes nothing to the result. On the right-hand side of the expression, we row-sample $A_{\mathbf{x}}$ by 2 and then multiply the transpose of that with $e'_{\mathbf{a}}$, which implicitly ignores every second row of the original $e_{\mathbf{a}}$ and uses every second column of $A_{\mathbf{x}}^T$ just like the left-hand side. An alternative proof is shown below:

$a(\mathbf{x}) = \mathbf{x} *_v \mathbf{k} + b$

$\sigma_r(a(\mathbf{x}), 2) = \sigma_r(\mathbf{x} *_v \mathbf{k} + b, 2)$

This equivalence is apparent from drawing out the row-sampled versions of the matrices from part b.

$\sigma_r(a(\mathbf{x}), 2) = \sigma_r(C_v(\mathbf{k})\mathbf{x} + b, 2)$

Differentiate both sides, which is still possible because they're matrices:

$\sigma_r(A_{\mathbf{x}}, 2) = \sigma_r(C_v(\mathbf{k}), 2)$

Using the equality from part b:

$\sigma_r(A_{\mathbf{x}}, 2) = \sigma_r(A_{\mathbf{x}}, 2)$

Problem 3(a)

```
function [y, x, a] = cnn(x1, net)

ok(net);

L = size(net, 1);

x = cell(L, 1);
a = cell(L, 1);

for i = 1:L
    x{i} = x1;
    curNet = net(i);
    curKernels = curNet.kernel;
    output = [];
    for j = 1:size(curKernels, 3)
        output = [output, convn(x1, curKernels(:, :, j), 'valid') + curNet.bias(j)];
    end
    x1 = output(1:curNet.stride:end, :);
    a{i} = x1;
    [x1, dhda] = curNet.h(x1);
```

```
end

y = x1;

end
```

$y = [534.6887, 130.5719, 347.6032, 0.1243, 759.6464]$
$e = 1.2245$

Problem 3(b)

```
function [g, e] = backprop(net, loss, xn, yn)

[y, x, a] = cnn(xn, net);

L = size(net, 1);

[e, ey] = loss(yn, y);

g = [];

for l=L:-1:1
    curNet = net(l);
    curKernel = curNet.kernel;
    [~, dhda] = curNet.h(a{l});

    ex = [];
    for j = size(curKernel, 3):-1:1
        m = size(x{l}, 1);
        n = size(curKernel, 1);
        p = m - n + 1;

        ea = ey(:, j) .* dhda(:, j);
        dea = dilute(ea, p);

        exComponent = convn(dea, reverse(curKernel(:, :, j)), 'full');
        if isempty(ex)
            ex = exComponent;
        else
            ex = ex + exComponent;
        end
        convResult = convn(dilute(ea, p), reverse(x{l}), 'full');
        ek = middle(convResult, n);
        eb = sum(ea);
        g = [g;ek(:);eb];
    end
    ey = ex;
end
```

```
end
```

Problem 3(c)

```
function [g, e] = numeric(net, loss, xn, yn)

d = 10^-6;

w1 = weights(net);

[y, x, a] = cnn(xn, net);
[e, ew] = loss(yn, y);

g = zeros(size(w1));

for i = 1:size(w1)
    di = zeros(size(w1));
    di(i) = d;

    wTemp = w1 + di;
    net = setWeights(wTemp, net);
    [y, ~, ~] = cnn(xn, net);
    [eTemp, ~] = loss(yn, y);

    g(i) = (eTemp - e)/d;
end


end
```

Problem 3(d)

```
function [e, ey]= ee2(yn, y)

e = norm(yn-y)^2;

d = 10^-6;

ey = zeros(size(y));

for i = 1:size(ey, 1)
    di = zeros(size(y));
    di(i) = d;
    yd = y + di;
    ey(i) = (norm(yn - yd)^2 - e) / d;
end
```

```
end
```