Backward Pass

In this part of the lab, we will implement a neural net's capability to propagate the error gradients from the last layer all the way to the first layer. We will refer to this procedure as a backward pass inside a neural network. At every fully connected layer l, we will need to do four things:

1. Compute the derivative of a non-linear sigmoid function $f(z) = \frac{1}{1+\exp(-z)}$, which can be computed as $\frac{\partial f(z)}{\partial z} = f(z)(1 - f(z))$. Note that your implementation needs to handle multi-dimensional vector inputs (i.e. use '.*' operator in matlab instead of '*'). The computed result $\frac{\partial f(z)}{\partial z}$ corresponding to layer l should be stored into nn.gradSigm{l} variable.
2. Use the error gradients $\frac{\partial L}{\partial z^{(l+1)}}$ coming from layer $l+1$ to compute the error gradients $\frac{\partial L}{\partial f(z^{(l)})}$ at the current layer l. This can be done as: $\frac{\partial L}{\partial f(z^{(l)})} = (W^{(l)})^T \frac{\partial L}{\partial z^{(l+1)}}$ where $(W^{(l)})^T$ depicts the transposed parameter matrix from layer l. The computed result $\frac{\partial L}{\partial f(z^{(l)})}$ should be stored into nn.gradA{l} variable.
3. Use the previously computed quantities $\frac{\partial L}{\partial f(z^{(l)})}$ and $\frac{\partial f(z)}{\partial z}$ to compute $\frac{\partial L}{\partial z^{(l)}} = \frac{\partial L}{\partial f(z^{(l)})} \odot \frac{\partial f(z^{(l)})}{\partial z^{(l)}}$ where $\odot$ is an elementwise multiplication implemented as '.*' in matlab. The computed quantity $\frac{\partial L}{\partial z^{(l)}}$ should be stored into the variable nn.gradZ{l} for every layer l in the network.
4. Finally, we need to compute the error gradients with respect to the fully connected layer parameters: $\frac{\partial L}{\partial W^{(l)}}$ for every layer l. This can be done as: $\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial z^{(l+1)}} (a^{(l)})^T$ where $(a^{(l)})^T$ denotes the transposed activation units from layer l. Note that $a^{(l)} = f(z^l)$. Then, the computed result $\frac{\partial L}{\partial W^{(l)}}$ should be stored into the variable nn.gradW{l} for every layer l in the network.

Your Function

Save Reset MATLAB Documentation (https://www.mathworks.com/help/)

```
function nn = BackwardPass(nn,y)
    % perform the forward pass inside the network
    %
    % Input:
    % - nn: a structure storing the parameters of the network
    % - y: a ground truth indicator matrix, where y(i,j)=1 indicates that a data point i belongs to an object c
    % Output:
    % - nn: a new neural network variable where the values nn.gradZ{l} and nn.gradW{l} are updated for every la


    %number of layers in a network
    n_layers=numel(nn.W)+1;

    %computing the overall error of the network
    error= nn.a{n_layers} - y';

    %computing the error gradient in the last layer of the network
    nn.gradZ{n_layers} = error .* (nn.a{n_layers} .* (1 - nn.a{n_layers}));

    %looping through the layers backwards
```

Code to call your function

Reset

```
architecture=[336 100 20];
nn = InitializeNetwork(architecture);
random_feature=rand(1,336);
random_label=rand(1,20);
nn = ForwardPass(nn, random_feature);
nn = BackwardPass(nn, random_label);
```

Run Function

Previous Assessment: All Tests Passed

Submit

Is the First Step of the Backward Pass Implemented Correctly?

Is the Second Step of the Backward Pass Implemented Correctly?

Is the Third Step of the Backward Pass Implemented Correctly?

Is the Fourth Step of the Backward Pass Implemented Correctly?