1.2 Regression Task

a 1. Forward Pass: Compute $g(\hat{\mathbf{y}} = Linear_2(f(Linear_1(\mathbf{x}))))$.

2. Evaluate Loss: Compute $\ell_{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2$

3. Clean Gradient: Programatically zero out the gradients. Ie. The computed gradients on each node of the computational graph should be set to 0.

4. Compute the gradient of the cost function with respect to the weights and bias using backpropogation. In this case that would be, $\frac{\partial \ell}{\partial \mathbf{W}^{(2)}}, \frac{\partial \ell}{\partial \mathbf{b}^{(2)}}$ and $\frac{\partial \ell}{\partial \mathbf{W}^{(1)}}, \frac{\partial \ell}{\partial \mathbf{b}^{(1)}}$

5. Adjust the weights and bias according to $\mathbf{W}^{(i)}_{(t+1)} = \mathbf{W}^{(i)}_{(t)} - \eta \frac{\partial \ell}{\partial \mathbf{W^{(i)}}_{(t)}}$ or $\mathbf{b}^{(i)}_{(t+1)} = \mathbf{b}^{(i)}_{(t)} - \eta \frac{\partial \ell}{\partial \mathbf{b^{(i)}}_{(t)}}$. Here t is the step number and η is the learning rate.

b Layer 1: Output: $\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$

Layer 1 Non-Linearity Output: $\text{relu}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$

Layer 2: $\mathbf{W}^{(2)}(\text{relu}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})) + \mathbf{b}^{(2)}$

Layer 2 Non-Linearity: $\mathbf{W}^{(2)}\text{relu}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}$

Cost: $\|(\mathbf{W}^{(2)}\text{relu}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) - \mathbf{y}\|^2$

c The gradients we are interested in are $\frac{\partial \ell}{\partial \mathbf{W}^{(1)}}, \frac{\partial \ell}{\partial \mathbf{b}^{(1)}}, \frac{\partial \ell}{\partial \mathbf{W}^{(2)}}, \frac{\partial \ell}{\partial \mathbf{b}^{(2)}}$. We will note that $\mathbf{x} \in \mathbb{R}^n$ and $\hat{\mathbf{y}} \in \mathbb{R}^K$

$$\frac{\partial \ell}{\partial \mathbf{b}^{(2)}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} \frac{\partial \mathbf{z_3}}{\partial \mathbf{b}^{(2)}}$$

.

Here $\frac{\partial \mathbf{z_3}}{\partial \mathbf{b}^{(2)}} = \mathbf{I} \in \mathbb{R}^{K \times K}$ so we have,

$$\frac{\partial \ell}{\partial \mathbf{b}^{(2)}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{z_3}}} \in \mathbb{R}^{1 \times K}$$

For $\frac{\partial \ell}{\partial \mathbf{W}^{(2)}}$ we have a similar procedure.

$$\frac{\partial \ell}{\partial \mathbf{W}^{(2)}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} \frac{\partial \mathbf{z_3}}{\partial \mathbf{W}^{(2)}} = \mathbf{z_2} \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{z_3}}} \in \mathbb{R}^{m \times k}$$

Now going down another layer.

$$\frac{\partial \ell}{\partial \mathbf{b}^{(1)}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} \frac{\partial \mathbf{z_3}}{\partial \mathbf{z_2}} \frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}} \frac{\partial \mathbf{z_1}}{\partial \mathbf{b}^{(1)}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} \mathbf{W^{(2)}} \frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}} \in \mathbb{R}^{1 \times m}$$

Here $\frac{\partial \mathbf{z_1}}{\partial \mathbf{b}^{(1)}} = \mathbf{I} \in \mathbb{R}^{m \times m}$. Similarly for $\mathbf{W}^{(1)}$ we have,

$$\frac{\partial \ell}{\partial \mathbf{W}^{(1)}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\frac{\partial \mathbf{z_3}}{\partial \mathbf{z_2}}\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}\frac{\partial \mathbf{z_1}}{\partial \mathbf{W}^{(1)}} = \mathbf{x}\frac{\partial \ell}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\mathbf{W^{(2)}}\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}} \in \mathbb{R}^{n\times m}$$

d $(\frac{\partial \ell}{\partial \hat{\mathbf{y}}})_i = 2(\hat{y}_i - y_i)$, ${\frac{\partial \ell}{\partial \hat{\mathbf{y}}}}_i \in \mathbb{R}^{1\times K}$

$\frac{\partial \hat{\mathbf{y}}}{\mathbf{z_3}} = \mathbf{I} \in \mathbb{R}^{K\times K}$. That is a $K \times K$ identity matrix.

$(\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}})_{i,i} = \begin{cases} 1 & \mathrm{f}(\mathrm{z}_{1i}) > 0 \\ 0 & \mathrm{f}(\mathrm{z}_{1i}) \le 0 \end{cases} \in \mathbb{R}^{m\times m}$

If the output features of the first layer are of dimension m.

Problem 1.3

Classification Task

a (a) For the forward pass you simply need to change the activation functions in the equations.
Layer 2 Output

$$\sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

Layer 3 Output

$$\sigma(W^{(2)}\sigma(W^{(1)}\mathbf{x} + (\mathbf{1})) + \mathbf{b^{(2)}})$$

(b) Our gradients are now going to be modified at the activation functions between layers. The term $\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = \sigma(\mathbf{z_3})(1 - \sigma(\mathbf{z_3}))\mathbf{I}$ and similarly $\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}} = \sigma(\mathbf{z_1})(1 - \sigma(\mathbf{z_1}))\mathbf{I}$

(c) $(\frac{\partial \ell}{\partial \hat{\mathbf{y}}})_i = 2(\hat{y}_i - y_i)$, ${\frac{\partial \ell}{\partial \hat{\mathbf{y}}}}_i \in \mathbb{R}^{1\times K}$
$\frac{\partial \hat{\mathbf{y}}}{\mathbf{z_3}} = \sigma(\mathbf{z_3})(1 - \sigma(\mathbf{z_3})\mathbf{I}) \in \mathbb{R}^{K\times K}$
$\frac{\partial \mathbf{z_2}}{\mathbf{z_1}} = \sigma(\mathbf{z_1})((1 - \sigma(\mathbf{z_1})\mathbf{I}) \in \mathbb{R}^{m\times m}$. If the output features of the first layer are of dimension m.

b Having multiple sigmoidal layers can lead to a problem of vanishing gradients. This is best seen through the derivative of the sigmoid function

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Note that $\sigma(x) \in (0, 1)$. So when computing the gradients on the backwards pass for each sigmoid function we are computing the product of two floats that are less that 1 and greater than 0. With multiple of these layers we start to see problems with respect the gradients of the network disapearing to to the repeated multiplication of values $\in (0, 1)$.

Problem 1.4

1. softmax does a sort of continuous version of argmax. In the traditional argmax, the maximum element of the vector is assigned a value of 1 while everything else is assigned to 0. In this sense argmax can still be considered a probability distribution as all of the elements sum to one. However most of the time we would like some degree of uncertainty in our probability distrubition, this is what is introduced with softmax, where we still get the maximum element being assigned the highest value, but the other elements in the vector are being assigned values in line with how big they are. Thus we now have a probability distribution with the same crucial property of argmax, namely that its maximum is given by the values of the array, but there is a greater degree of knowledge about the uncertainty around this value.

2. The softmax function naturally includes a lot of exponentials of variables in its computation. This has the potential to create problems when we are working with extremely high and extremely low values. This can lead to overflow and underflow errors in our computations.

3. No we should not have two consecutive linear layers in a neural network. The reason being is that they are redundant and can be perfectly represented by a single linear layer. Take for instance an input $x \in \mathbb{R}^n$ and two linear layers, disregarding the bias for simplicity, with $\mathbf{W^{(1)}} \in \mathbb{R}^{m \times n}, \mathbf{W^{(2)}} \in \mathbb{R}^{k \times m}$. The sequence of layers would look like,

   $\mathbf{W^{(2)}W^{(1)}x}$ and the output would be in $\mathbb{R}^{k \times n}$. This is exactly equivalent to a single linear layer with weight matrix equal to the product of $\mathbf{W^{(2)}W^{(1)}}$

4. ReLU:

   Advantage- You do not run into the vanishing gradient problem that you would run into with the sigmoid function, and it is computationally very efficient as you are just selecting an element from a vector or making it 0.

   Disadvantage- It is possible with the ReLU to have a problem if all of your inputs are negative, then you will get just a matrix/vector of zeroes and similarly a gradient of zeroes rendering the non-linearity essentially meaningless. Similarly if you have huge values in your input they will get mapped to the same huge values.

   Tanh:

   Advantage- This allows us to have a sigmoidal function that will include negative outputs for negative inputs.

   Disadvantage- Tanh will still suffer from the problem of vanishing gradients as the values are bounded between (-1,1).

   Sigmoid:

   Advantage-Good for binary classification, and keeps activation function from growing very large.

   Disadvantage- Vanishing gradients make it hard to use.

Leaky ReLU:

Advantage: Deals with the dying neuron problem of ReLU by allowing negative value to get mapped to other negative values based on a predetermined slope for a linear function on negative inputs.

Disadvantage- Obviously still faces the same problem of activation fucntions blowing up that traditional ReLU faces.

5. 1. Rotation
   2. Scaling
   3. Projection
   4. Reflection

   The linear transformations in a neural network are what give us the ability to adjust what is going on with respect to some kind of complexity measure while simultaneously giving us access to more dimensionality and therefore flexibility in our network. The non-linear transformations on the other hand give us a much richer class of functions that we can approximate by allowing non-linear outputs from layers of our network.

6. As batch size goes up we are getting a gradient closer to the ”True” gradient so we can allow our learning rate to be bigger. With a smaller batch size a smaller learning rate makes sens because the approximation of the true gradient is going to be less accurate and we want to leverage this stochasticity by making more small adjustments.