Due Date: October 19th, 2023 at 11:00 pm

Question 1 (5-2-2-1-3-2)**.** (**Activation Functions and Backpropagation**) *Please note that during marking we will not read more than the question says you should write, e.g. if the question asks for one sentence, only the first sentence will be read.*

1. Given the following 3 layer neural network $\boldsymbol{o} = f(\boldsymbol{x})$

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1 \\ \boldsymbol{a} &= \sigma(\boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2) \\ \boldsymbol{o} &= \boldsymbol{W}_3\boldsymbol{a} + \boldsymbol{b}_3, \end{aligned} \tag{1}$$

   where $\boldsymbol{x} \in \mathrm{R}^n$, $\boldsymbol{h} \in \mathrm{R}^m$, $\boldsymbol{a} \in \mathrm{R}^r$, $\boldsymbol{o} \in \mathrm{R}^s$, $\sigma(x) = \frac{1}{1+e^{-x}}$, and a loss function $\mathrm{L}(\boldsymbol{o}, \boldsymbol{y})$ apply the back-propagation algorithm computing the gradients with respect to the weights $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3$ to show which are the necessary intermediate vectors that we need to store during the forward pass such as to not need recomputation. Use the notation provided above.
2. Assuming each floating point number takes two bytes to store, give the amount of memory in bytes, needed to store during the forward pass for the backward pass to not need recomputation.
3. Compute the left and right derivative of the `ReLU` function at $x = 0$. Show your work.
4. Give **one** sentence to explain why we cannot use the `ReLU` function if we want to differentiate our neural network twice.
5. The swish activation function is defined as $g(x) = x\sigma(\beta x)$. Simplify the function in the limit as β approaches 0 and infinity. Show your work.
6. Give 1 reason (one sentence) as to why we might want to use the swish function over the `ReLU`. Give 1 reason (one sentence) as to why we might want to use the `ReLU` over the swish (hint: monotonicity).

Answer 1. 1. - For the 3rd layer:

$$\frac{\partial L}{\partial W_3} = \frac{\partial o}{\partial W_3}\frac{\partial L}{\partial o} = a\frac{\partial L}{\partial o}$$
$$\frac{\partial L}{\partial b_3} = \frac{\partial o}{\partial b_3}\frac{\partial L}{\partial o} = \frac{\partial L}{\partial o}$$

- For the 2nd layer:
We know that:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$
$$\begin{aligned} \frac{\partial a}{\partial W_2} &= \frac{\partial W_2 h + b_2}{\partial W_2}\frac{\partial \sigma(W_2 h + b_2)}{\partial W_2 h + b_2} \quad \text{(chain rule)} \\ &= h \cdot a \cdot (1-a) \end{aligned}$$
$$\frac{\partial a}{\partial b_2} = a \cdot (1-a) \quad \text{(chain rule)}$$

So:

$$\frac{\partial L}{\partial W_2} = \frac{\partial a}{\partial W_2}\frac{\partial o}{\partial a}\frac{\partial L}{\partial o} = \frac{\partial L}{\partial o} \cdot W_3 \cdot h \cdot a \cdot (1-a)$$
$$\frac{\partial L}{\partial b_2} = \frac{\partial a}{\partial b_2}\frac{\partial o}{\partial a}\frac{\partial L}{\partial o} = \frac{\partial L}{\partial o} \cdot W_3 \cdot a \cdot (1-a)$$

- For the 1st layer:

$$\frac{\partial L}{\partial W_1} = \frac{\partial h}{\partial W_1}\frac{\partial a}{\partial h}\frac{\partial L}{\partial o} = x\frac{\partial L}{\partial h}$$
$$\frac{\partial L}{\partial b_1} = \frac{\partial h}{\partial b_1}\frac{\partial a}{\partial h}\frac{\partial L}{\partial o} = \frac{\partial L}{\partial h}$$

With:

$$\begin{aligned}\frac{\partial L}{\partial h} &= \frac{\partial a}{\partial h}\frac{\partial L}{\partial a} \\ &= W_2 a(1-a)\frac{\partial L}{\partial a}\end{aligned}$$

and

$$\begin{aligned}\frac{\partial L}{\partial a} &= \frac{\partial o}{\partial a}\frac{\partial L}{\partial o} \\ &= W_3 a(1-a)\frac{\partial L}{\partial o}\end{aligned}$$

We observe that everything can be expressed in function of $\frac{\partial L}{\partial o}$ along with h and a (we consider x already being stored and not intermediate since it's the input vector).

2. We have $h \in \mathbb{R}^m$, $a \in \mathbb{R}^r$ and $\frac{\partial L}{\partial o} \in \mathbb{R}^s$. So we need to store vectors of sizes m, r and s.

$$2(m+r+s) \quad \text{Bytes}$$

3.

$$\text{ReLU} = f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$
$$\text{Left: } \lim_{x \to 0^-} \frac{f(x) - f(0^-)}{x - 0^-} = \frac{0-0}{x} = 0$$
$$\text{Right: } \lim_{x \to 0^+} \frac{f(x) - f(0^+)}{x - 0^+} = \frac{x-0}{x} = 1$$

So:

$$\text{ReLU}(x)' = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

With $x = 0$ undefined.

4. With ReLU(0), when we differentiate our neural network twice and therefore derive the activation function twice, we will have 0 values everywhere except at x=0, where it is undefined, and so it's pretty much useless.
5. We have:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\lim_{x\to 0} \sigma(x) = \frac{1}{2}$$
$$\lim_{x\to +\infty} \sigma(x) = 1$$
$$\lim_{x\to -\infty} \sigma(x) = 0$$

So with that we can calculate:

$$\lim_{\beta\to 0} g(x) = \frac{1}{2}x$$

Since we don't know if $x > 0$ or $x < 0$

$$\lim_{\beta\to \infty} g(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

6. We might want to use the swish function over the ReLU because we can fully differentiate our neural network with it, without having a non defined point and we can fully differentiate our neural network twice.
We might want to use the ReLU over the swish because it's monotonically increasing and therefore the gradients are less likely to change signs randomly allowing a more stable convergence during training.

Question 2 (4-4-5-3)**.** (**Convolution Basics**)

1. **Short answers:**
   (a) When you apply a filter to feature maps, under what conditions does the size of the feature map decrease ?
   (b) What impact does a larger stride have ?
   (c) Explain what you understand from the term "sparse connections" in terms of connections between two layers in a neural network.
   (d) As we are aware, each *Convolutional* layer contains learnable parameters, including weights and biases. If you decide not to include biases in the *Convolution* layer of the following network, what will be the impact on its performance (compared to the case when the *Convolution* layer includes biases) ?

$$Input \rightarrow Convolution \rightarrow BatchNorm \rightarrow Activation \rightarrow Output$$

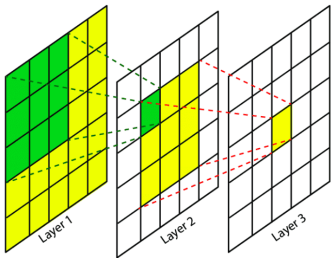

FIGURE 1 – Illustration of Receptive Field.

2. **True/False:**
   (a) In a convolutional layer, the number of weights is contingent upon the depth of the input data volume.
   (b) In a convolutional layer, the number of biases equals the number of filters.
   (c) The total number of parameters is influenced by the stride and padding.
   (d) Maxpooling operates only on the height and width dimensions of an image.
3. **Convolutional Architecture:** Let's analyze the convolutional neural network (CNN) defined by the layers in the left column. For each layer, determine the output volume's shape and the number of parameters. It's worth noting that in our notation, "Conv4-N-Pvalid-S2" represents a convolutional layer with N 4x4 filters, valid padding, and a stride of 2.

Layers	Output volume dimensions	Number of parameters
Input	$64 \times 64 \times 1$	
Conv4-5-Pvalid-S2		
Pool3		
Conv3-5-Pvalid-S1		
Pool2		
FC5		

4. **Receptive Fields:** Receptive Field is defined as the size of the region in the input that produces a feature. For example, when using a Conv-3x3 layer, each feature is generated by a 3x3 grid in the input. Having two Conv-3x3 layers in succession leads to a 5x5 receptive field. An example of this is outlined in Figure 1. Note that there is a receptive field associated with each feature (Figure 1 shows the receptive field associated with the yellow-colored feature in Layer 3), where a feature is a cell in the output activation map.

   Consider a network architecture consisting solely of 5 Convolution layers, each with kernel size 5, stride 1, zero-padding of two pixels on all four sides, 8 intermediate channels, and one output channel. What is the receptive field of the final features obtained through this architecture, that is, what is the size of the input that corresponds to a particular pixel position in the output ?

Answer 2. 1. (a) If there is no padding and a stride of at least 1

(b) Larger stride creates smaller deature map outputs and lowers the model capacity

(c) Each neuron in a layer is connected to a subset of the next layer. Just like in CNNs where each bloc/unit in a feature map is connected only to a small region of pixels of the input image.

(d) Wouldn't normalizing just ignore the effects of the bias ? So basically with or without bias has no impact with batch normalization.

2. (a) True.

(b) True.

(c) False. Number of parameters $= (n * m * l + 1) * k$
Where $n * m$ is the size of the filter
l number of feature maps input k number of feature maps output

(d) True

3. W = Wdith, H = height, D = depth, F = filtersize, N = nb Filters, S = stride
Input:

$$W_1 = 64$$
$$H_1 = 64$$
$$D_1 = 1$$
$$\text{Parameters} = 0$$

Conv4-5-Pvalid-S2:

$$F_1 = 4$$
$$N_1 = 5$$
$$S_1 = 2$$

Conv3-5-Pvalid-S1:

$$F_2 = 3$$
$$N_2 = 5$$
$$S_2 = 1$$

Layer 2: 31x31x5

$$W_2 = \lfloor \frac{W_1 - F_1}{S_1} \rfloor + 1 = 31$$
$$H_2 = \lfloor \frac{H_1 - F_1}{S_1} \rfloor + 1 = 31$$
$$D_2 = N_1 = 5$$
$$\text{Parameters: } D_1 \times F_1 \times F_1 \times N_1 + N_1 = 85$$

Layer 3: 10x10x5

$$W_3 = \lfloor \frac{W_2}{3} \rfloor = 10$$
$$H_3 = \lfloor \frac{H_2}{3} \rfloor = 10$$
$$D_3 = D_2 = 5$$
$$\text{Parameters} = 0$$

Layer 4: 8x8x5

$$W_4 = \lfloor \frac{W_3 - F_2}{S_2} \rfloor + 1 = 8$$
$$H_4 = \lfloor \frac{H_3 - F_2}{S_2} \rfloor + 1 = 8$$
$$D_4 = 5$$
$$\text{Parameters: } D_3 \times F_2 \times F_2 \times N_2 + N_2 = 230$$

Layer 5: 4x4x5

$$W_5 = \lfloor \frac{W_4}{2} \rfloor = 4$$
$$H_5 = \lfloor \frac{H_4}{2} \rfloor = 4$$
$$D_5 = 5$$
$$\text{Parameters} = 0$$

Layer 6: 1x1x5

$$W_6 = 1$$
$$H_6 = 1$$
$$D_6 = 5$$
$$\text{Parameters} = D_5 \times W_5 \times H_5 \times FC5 + FC5 = 405$$

4. The formula for receptive layer sizes is:

$$R_l = R_{l-1} + (k_l - 1) \times \prod_{i=1}^{l-1} s_i$$

So we can compute the layer sizes as follow (don't know why table isn't blue too):

Layer	Receptive field size
1	5 x 5
2	$5 + (5-1) \times 1 = 9 \rightarrow$ 9 x 9
3	$9 + (5-1) \times 1^2 = 13 \rightarrow$ 13 x 13
4	$13 + (5-1) \times 1^3 = 17 \rightarrow$ 17 x 17
5	$17 + (5-1) \times 1^4 = 21 \rightarrow$ 21 x 21

Question 3 (10). (**Mixture Models meet Neural Networks**)

Consider modelling some data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^d$, $y_n \in \{0, 1\}$, using a mixture of logistic regression models, where we model each binary label y_n by first picking one of the K logistic regression models, based on the value of a latent variable $z_n \sim$ Categorical$(\pi_1, ..., \pi_K)$ and then

generating y_n *conditioned* on z_n as $y_n \sim \text{Bernoulli}[\sigma(\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n)]$, where $\sigma(\cdot)$ is the sigmoid activation function.

Now consider the *marginal* probability of the label $y_n = 1$, given $\boldsymbol{x}_n$, i.e., $p(y_n = 1|\boldsymbol{x}_n)$, and show that this quantity can also be thought of as the output of a neural network. Clearly specify what is the input layer, the hidden layer(s), activations, the output layer, and the connection weights of this neural network.

Answer 3.

$$P(y_n = 1|x_n) = \sum_{k=1}^{K} P(y_n = 1|z_n = k|x_n)$$

We have:

$$P(y_n = 1|z_n = k|x_n) = P(y_m = 1|z_n = k, x_n) \cdot P(z_n = k|x_n)$$

and

$$y_n|z_n = k, x_n \sim \text{Bernoulli}\left(\mathbb{E}\left[w_k^T x_n\right]\right)$$

z_n does not depend on x_n so $P(z_n = k|x_n) = P(z_n = k) = \pi_k$
Therefore:

$$P(y_n = 1|x_n) = \sum_{k=1}^{K} \sigma(w_k^T x_n) \cdot \pi_k$$

Input layer: The input layer consists of d neurons
Connection to next layer: Connecting the input layer and first are the K weights W
First hideen layer: Consists of K neurons (k^{th} neuron being a logistic regression). The activation of the k-th neuron in the layer is given by: $a_k = (w_k^T x_n)$
Connection to next layer: Connecting the first hidden layer and output layer are the π_K probabilities
Output layer: There is a single neuron at the ouput calculating the weighted sum of the activations from the first hideden layer. The ouput neuron has activation: $P(y_n = 1|x_n) = \sum_{k=1}^{K} a_k \cdot \pi_k$

Question 4 (10)**.** (**Cross Entropy**)

Cross-entropy loss function (a popular loss function) is given by:

$$ce(p, x) = -x \log(p) - (1 - x) \log(1 - p)$$

This derivation assumes that x is binary, i.e. $x \in \{0, 1\}$. However, the same loss function is often also used with real-valued $x \in (0, 1)$.

1. Derive the cross-entropy loss function using the maximum likelihood principle for $x \in \{0, 1\}$.
2. Suggest a probabilistic interpretation of the cross-entropy loss function when $x \in (0, 1)$. (HINT: KL divergence between two distributions)

Answer 4. 1.

$$\begin{aligned}
\mathcal{L}(\theta) &= \arg\max P(X|\theta) \\
&= \arg\max \prod_{i=1}^{n} P(x_i|\theta) \quad \text{because } x_i \text{ are independent} \\
&= \arg\max \log(\prod_{i=1}^{n} P(x_i|\theta)) \\
&= \arg\max \sum_{i=1}^{n} \log(P(x_i|\theta)) \\
&= \arg\min - \sum_{i=1}^{n} \log(P(x_i|\theta))
\end{aligned}$$

Bernoulli probability mass function:

$$P(x_n|p) = p^{x_n}(1-p)^{1-x_n}$$

We replace:

$$\begin{aligned}
-\log \mathcal{L}(p) &= -\sum_{i=1}^{m} \log(p^{x_n}(1-p)^{1-x_n}) \\
&= -\sum_{i=1}^{n} (x_n \log(p) + (1-x_n)\log(1-p)) \\
&= -(x \log(p) + (1-x_n)\log(1-p)) \quad \text{For a point} \qquad = ce(p, x)
\end{aligned}$$

2. $D_{\mathrm{KL}}(P||Q) = \sum_i P(x_i)(\log P(x_i) - \log Q(x_i))$
Let P be the correct distribution and Q be the predicted distribution.

$$\begin{aligned}
P(1) &= x \\
P(0) &= 1 - x \\
Q(1) &= p \\
Q(0) &= 1 - p
\end{aligned}$$

$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= (1-x)\log(\frac{1-x}{1-p}) + x\log(\frac{x}{p}) \\ &= -xlog(p) - (1-x)log(1-p) + xlog(x) + (1-x)log(1-x) \\ &= ce(p,x) + xlog(x) + (1-x)log(1-x) \end{aligned}$$

$xlog(x) + (1-x)log(1-x)$ is the entropy of the true distribution P which measures its uncertainty.
While $ce(p,x)$ calculates how far the predictions p are from our labels x.

The cross-entropy can be seen as the additional uncertainty introduced when we use our predicted distribution Q to approximate the true distribution P. Therefore, when $x \in (0,1)$, the cross-entropy loss is the difference between the true distribution's entropy and its KL divergence with the predicted distribution.
I guess we could say, it sort of measures the "loss" of using Q to represent P.

Question 5 (1-4-2)**.** (**Optimization**)

Suppose we want to minimize $f(x,y) = y^2 + (y-x)^2$.

1. Find the true minimum.
2. For each step size given below, compute:
   (a) The value of (x1, y1) using the full gradient descent (not stochastic) at the starting point (x0, y0) = (1, 1).
   (b) The L2 distance of (x1, y1) from the true minimum.

   Please note that the (x1, y1) is (x, y) coordinate after 1 step of gradient descent.

   i. Step size $s = 0.01$.
   ii. Step size $s = 0.1$.
   iii. Step size $s = 10$.
3. What are the implications of using different step sizes in gradient descent, and what strategies can be employed to effectively address these implications ?

Answer 5. 1.

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} &= -2(y-x) \\ \frac{\partial f(x,y)}{\partial y} &= 2y + 2(y-x) \end{aligned}$$

$$\begin{cases} -2(y-x) & = 0 \\ 2y + 2(y-x) & = 0 \end{cases}$$

$$\begin{cases} y = x \\ 4y = 2x \implies y = \frac{x}{2} \end{cases}$$

So we have $x = 0$ and $y = 0$ with true minimum at $(0, 0)$

2. (a) Gradients:

$$x_n = x_{n-1} - S(-2(y - x))$$
$$y_n = y_{n-1} - S(2y + 2(y - x))$$

S = 0.01:

$$x_1 = 1 - 0.01(-2(1 - 1)) = 1$$
$$y_1 = 1 - 0.01(2 + 0) = 0.98$$

S = 0.1

$$x_1 = 1 - 0.1(-2(1 - 1)) = 1$$
$$y_1 = 1 - 0.1(2 + 0) = 0.8$$

S = 10

$$x_1 = 1 - 10(-2(1 - 1)) = 1$$
$$y_1 = 1 - 10(2 + 0) = -19$$

(b) S = 0.01:

$$L_2 = \sqrt{(1 - 0)^2 + (0.98 - 0)^2} = 1.40$$

S = 0.1:

$$L_2 = \sqrt{(1 - 0)^2 + (0.8 - 0)^2} = 1.28$$

S = 10:

$$L_2 = \sqrt{(1 - 0)^2 + (-19 - 0)^2} = 19.03$$

3. We can see from our example, that having a big step can make us overshoot the minimum and very probably oscillate around it, leading to none convergence ($s = 10$). On the other hand, with a step that is too small, it will very slowly go towards the minimum, leading to extremely slow training ($s = 0.01$). Finding the correct step size, will lead to effecient convergence and minimize the risks of overshooting ($s = 0.1$).

Many strategies can been used to effectively optimize hyperparameter. These strategies often involve dynamically changing the hyperparameter through cycles. Here are 2 examples:

- We can use *momentum*: As the name of the strategy states, if the gradient encounters a "steep hill" we will update its step to push it forward and accelerate it, therefore increasing its speed of convergence. Then if it encounters small pits, the momentum gained will allow it not to get stuck there and search further.
- We can use *early stopping*: We evaluate our model on a validation set periodically in parallel to our training. Whenever the model's performance on the validation set starts to degrade or stagnate we halt the training process or reduce the learning rate. This prevents overfitting and reduces the training time if our training is going on for too long and not leading to any improvement.