Due Date : February 23rd (11pm), 2022

Instructions

- *For all questions, show your work!*
- *Use LaTeX when writing your answers. We recommend using this assignment latex code as a template. You may reuse most of the notation shorthands, equations and/or tables. See the assignment policy on the course website for more details.*
- *Submit your answers electronically via Gradescope.*
- ***TAs for this assignment are Matthew Scicluna and Akram Erraqabi.***

Question 1 (3-3-3-4-3-4-3)**.** Consider training a standard feed-forward neural network. For the purposes of this question we are interested in a single iteration of SGD on a single training example : $(\boldsymbol{x}, y)$. We denote $f(\boldsymbol{x}, \boldsymbol{\theta})$ as the output of the neural network with model parameters $\boldsymbol{\theta}$. Now let's say g is the output activation function and $a(\boldsymbol{x}, \boldsymbol{\theta})$ is the pre-activation network output such that $f(\boldsymbol{x}, \boldsymbol{\theta}) = g(a(\boldsymbol{x}, \boldsymbol{\theta}))$.

1.1 Assuming the network's goal is to do binary classification (with the detailed structure above), what would be an appropriate activation function for the output layer, i.e. what would be an appropriate function g ? *We will keep this choice of g for the rest of this exercise.*

answer :The sigmoid function is widely used for binary classification problems. It returns a value between 0 and 1, which can be considered as probabilities of belonging our input to a specific class.

1.2 What does the output represent under this activation function ?

answer :The output of Sigmoid function will always be between 0 and 1 and the equation is :

$$g(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

1.3 Let $L_{CE}(f(\boldsymbol{x}, \boldsymbol{\theta}), y)$ be cross-entropy loss, express it as a function of $f(\boldsymbol{x}, \boldsymbol{\theta})$ and y.

answer : M : number of classes
if $M = 2$ (binary cassificetion)

$$L_{CE}(f(\boldsymbol{x}, \boldsymbol{\theta}), y) = -(y \log(f(\boldsymbol{x}, \boldsymbol{\theta})) + (1 - y) \log(1 - f(\boldsymbol{x}, \boldsymbol{\theta}))$$

if $M > 2$ (multi class classification)

$$L_{CE}(f(x, \theta), y) = -\sum_{C=1}^{M} y_{o,c} \log\left(f_{o,c}(x, \theta)\right)$$

1.4 Compute the partial derivative $\frac{\partial L_{CE}(f(\boldsymbol{x},\boldsymbol{\theta}),y)}{\partial a(\boldsymbol{x},\boldsymbol{\theta})}$.

answer : We know that : $f(x,\theta) = g(a(x,\theta)) = \frac{1}{1+e^{-(a(x,\theta))}}$

$$\frac{\partial L_{CE}(f(x,\theta),y)}{\partial a(x,\theta)} = \frac{\partial L_{CE}(f(x,\theta),y)}{\partial f(x,\theta)} \times \frac{\partial f(x,\theta)}{\partial a(x,\theta)}$$
$$= \left(\frac{-y}{f(x,\theta)} + \frac{1-y}{1-f(x,\theta)}\right) \times \left(\frac{e^{-a(x,\theta)}}{\left(1+e^{-a(x,\theta)}\right)^2}\right)$$

1.5 Let $L_{MSE}(f(\boldsymbol{x},\boldsymbol{\theta}),y)$ be the mean-squared error, express it as a function of $f(\boldsymbol{x},\boldsymbol{\theta})$ and y (multiplicative factors can be ignored).

answer :

$$L_{\text{MSE}}\left(f(x,\theta),y\right) = (y - f(x,\theta))^2 \tag{2}$$

1.6 Compute the partial derivative $\frac{\partial L_{MSE}(f(\boldsymbol{x},\boldsymbol{\theta}),y)}{\partial a(\boldsymbol{x},\boldsymbol{\theta})}$.

answer :

$$\frac{\partial L_{\text{MSE}}\left(f(x,\theta),y\right)}{\partial a(x,\theta)} = \frac{\partial L_{\text{MSE}}\left(f(x,\theta),y\right)}{\partial f(x,\theta)} \times \frac{\partial f(x,\theta)}{\partial a(x,\theta)}.$$
$$= -2(y - f(x,\theta)) \times \left(\frac{e^{-a(x,\theta)}}{\left(1+e^{-a(x,\theta)}\right)^2}\right) \tag{3}$$

1.7 Based on your answers to the above questions, what would be the more appropriate loss function for binary classification and why ?

answer : Cross-Entropy is a more appropriate loss for binary classification because :
1) consider we use the sigmoid function as our activation function, using MSE as loss function will converge slower than cross-entropy because of gradient vanishing issue.
2)cross-entropy is better because the decision boundary in classification is large, and MSE does not punish misclassification enough compared to cross-entropy.
3)Suppose you use a sigmoid as an activation layer in your neural network. In that case, it's natural to use cross-entropy as your cost function to maximize the likelihood of correctly classifying the input data. and if the output data is continuous and normally distributed, it's better to use MSE. [1] 4) MSE function is non-convex for binary classification. So it will not be guaranteed to minimize the lost function during training. 5) using MSE as loss function means that we assume that our data comes from a normal distribution but in binary classification our data comes from a Bernoulli distribution.

Question 2 (4-4-5-6)**.** Recall the definition of the softmax function : $S(\boldsymbol{x})_i = e^{\boldsymbol{x}_i}/\sum_j e^{\boldsymbol{x}_j}$.

2.1 Show that softmax is translation-invariant, that is : $S(\boldsymbol{x}+c) = S(\boldsymbol{x})$, where c is a scalar constant.

answer :

$$S(x+c)_i = \frac{e^{(x_i+c)}}{\sum_j e^{\left(x_j+c\right)}} = \frac{e^{x_i}e^c}{\sum_j e^{x_j}e^c}$$
$$= \frac{e^{x_i}}{\sum_j e^{x_j}} = S(x)_i$$

2.2 Let $\boldsymbol{x}$ be a 2-dimensional vector. One can represent a 2-class categorical probability using softmax $S(\boldsymbol{x})$. Show that $S(\boldsymbol{x})$ can be reparameterized using sigmoid function, i.e. $S(\boldsymbol{x}) = [\sigma(z), 1-\sigma(z)]^\top$ where z is a scalar function of $\boldsymbol{x}$.

answer :

$$\begin{cases} S(x)_1 = \dfrac{e^{x_1}}{e^{x_1}+e^{x_2}} \overset{\div e^{x_1}}{\Rightarrow} \dfrac{1}{1+e^{x_2-x_1}} \overset{z=x_2-x_1}{\Rightarrow} \dfrac{1}{1+e^{z}} = \sigma(z) \\ S(x)_2 = \dfrac{e^{x_2}}{e^{x_1}+e^{x_2}} \overset{\div e^{x_2}}{\Rightarrow} \dfrac{1}{1+e^{-(x_2-x_1)}} \overset{z=x_2-x_1}{\Rightarrow} \dfrac{1}{1+e^{-z}} = \sigma(-z) = 1-\sigma(z) \end{cases}$$

$$S(X) = [S(X)_1, S(X)_2]^\top = [\sigma(z), 1-\sigma(z)]^\top$$

2.3 Let $\boldsymbol{x}$ be a K-dimensional vector ($K \geq 2$). Show that $S(\boldsymbol{x})$ can be represented using $K-1$ parameters, i.e. $S(\boldsymbol{x}) = S([0, y_1, y_2, ..., y_{K-1}]^\top)$, where y_i is a scalar function of $\boldsymbol{x}$ for $i \in \{1, ..., K-1\}$.

answer : x is K-dimensional so we can write :

$$S(x) = S\left([x_1, x_2, \dots, x_k]^\top\right)$$

as we prove that softmax is translation-invarient So we can write :

$$S(x) = S(x - x_1) = S([x_1 - x_1, x_2 - x_1, \dots, x_k - x_1])$$

So we can consider $x_{i+1} - x_1 = y_i$ where ie $\{1, \dots, k-1\}$ we Can conclude that :

$$S(x) = S\left([0, y_1, y_2, \cdots, y_{k-1}]^\top\right)$$

2.4 Show that the Jacobian of the softmax function $J_{\text{softmax}}(\boldsymbol{x})$ can be expressed as : $\mathbf{Diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$, where $\boldsymbol{p} = S(\boldsymbol{x})$.

answer :

$$J_{\text{softmax}}(x) = \begin{pmatrix} \frac{\partial s_1}{\partial x_1} & \frac{\partial s_1}{\partial x_2} & \cdots & \frac{\partial s_1}{\partial x_k} \\ \frac{\partial s_2}{\partial x_1} & \frac{\partial s_2}{\partial x_2} & \cdots & \frac{\partial s_n}{\partial x_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s_n}{\partial x_1} & \frac{\partial s_n}{\partial x_2} & \cdots & \frac{\partial s_n}{\partial x_n} \end{pmatrix}$$

$$\frac{\partial s_i}{\partial x_j} = \begin{cases} \frac{e^{x_i}\left(\sum_j e^{x_j}\right) - e^{x_i}e^{x_i}}{\left(\sum_j e^{x_j}\right)^2} = \frac{e^{x_i}}{\sum_j e^{x_j}} - \frac{e^{x_i}e^{x_i}}{\sum e^{x_j}\sum_j e^{x_j}} = s_i - s_i^2 & i = j \\ \frac{-e^{x_j}e^{x_i}}{\left(\sum_j e^{x_j}\right)^2} - \frac{-e^{x_j}}{\sum_j e^{x_j}} \times \frac{e^{x_i}}{\sum_j e^{x_j}} = -s_i s_j & i \neq j \end{cases}$$

$$J_{\text{softmax}}(x) = \begin{pmatrix} S_1 - S_1 \cdot S_1 & -S_1 \cdot S_2 & \cdots & -S_1 \cdot S_k \\ -S_2 \cdot S_1 & S_2 - S_2 \cdot S_2 & \cdots & -S_2 \cdot S_k \\ \vdots & \vdots & \ddots & \vdots \\ -S_k \cdot S_1 & -S_k \cdot S_2 & \cdots & S_k - S_k \cdot S \end{pmatrix}$$

$$= \begin{pmatrix} s_1 & 0 & \cdots & 0 \\ & s_2 & \cdots & 0 \\ 0 & \vdots & \ddots & \vdots \\ \vdots & 0 & \cdots & s_k \end{pmatrix} - \begin{pmatrix} s_1 & s_2 & \cdots & s_k \\ s_1 & s_2 & \cdots & s_k \\ \vdots & \vdots & \ddots & \vdots \\ s_1 & s_2 & \cdots & s_k \end{pmatrix} \begin{pmatrix} s_1 & s_1 & \cdots & s_1 \\ s_2 & s_2 & \cdots & s_2 \\ \vdots & \vdots & \ddots & \vdots \\ s_k & s_k & \cdots & s_k \end{pmatrix} = \text{Diag}(p) - pp^\top$$

Question 3 (6)**.** Consider a 2-layer neural network $y : \mathbb{R}^D \to \mathbb{R}^K$ of the form :

$$y(x, \Theta, \sigma)_k = \sum_{j=1}^{M} \omega_{kj}^{(2)} \sigma \left(\sum_{i=1}^{D} \omega_{ji}^{(1)} x_i + \omega_{j0}^{(1)} \right) + \omega_{k0}^{(2)}$$

for $1 \leq k \leq K$, with parameters $\Theta = (\omega^{(1)}, \omega^{(2)})$ and logistic sigmoid activation function σ. Show that there exists an equivalent network of the same form, with parameters $\Theta' = (\tilde{\omega}^{(1)}, \tilde{\omega}^{(2)})$ and tanh activation function, such that $y(x, \Theta', \tanh) = y(x, \Theta, \sigma)$ for all $x \in \mathbb{R}^D$, and express Θ' as a function of Θ.

answer :

$$\begin{aligned} \tanh(x) &= \tfrac{e^x - e^{-x}}{e^x + e^{-x}} = \tfrac{e^x + e^{-x} - 2e^{-x}}{e^x + e^{-x}} = 1 + \tfrac{-2e^{-x}}{e^x + e^{-x}} \\ &= 1 - \tfrac{2}{e^{2x}+1} = 1 - 2\sigma(-2x) = 1 - 2(1 - \sigma(2x)) \\ &= 1 - 2 + 2\sigma(2x) = 26(2x) - 1 \end{aligned}$$

$$\begin{aligned} y\left(x, \theta', \tanh\right)_k &= \textstyle\sum_{j=1}^{1} \tilde{w}_{Kj}^{(2)} \left(2\sigma 2 \left(\sum_{i=1}^{D} \tilde{w}_{ji}^{(1)} x_i + \tilde{w}_{j0}^{(1)} \right) - 1 \right) + \tilde{w}_{k0}^{(2)} \\ &= \textstyle\sum_{j=1}^{M} 2\tilde{w}_{Kj}^{(2)} \sigma \left(\sum_{i=1}^{D} 2\tilde{w}_{ji}^{(1)} x_i + 2\tilde{w}_{j0}^{(1)} \right) - \sum_{j=1}^{M} \tilde{w}_{kj}^{(2)} + \tilde{w}_{k0}^{(2)} \end{aligned}$$

we can assume $y(x, \theta', \tanh) = y(x, \theta, \sigma)$ if :

$\tilde{w}_{kj}{}^{(2)} = \frac{1}{2} w_{kj}^{(2)} \quad j \in \{1, \ldots, M\}$

$\tilde{w}_{ji}^{(1)} = \frac{1}{2} w_{ji}^{(1)} \quad j \in \{1, \ldots, D\}$

$\tilde{w}_{j0} = \frac{1}{2} w_{j0}^{(1)} \quad j \in \{1, \ldots, D\}$

$\tilde{w}_{k0}^{(2)} = w_{k0}^{(2)} + \sum_{j=1}^{M} \tilde{w}_{kj}^{(2)} = w_{k0}^{(2)} + \sum_{j=1}^{M} \frac{1}{2} w_{kj}^{(2)}$

Question 4 (5-5-6)**.** Consider a convolutional neural network. Assume the input is a colorful image of size 128×128 in the RGB representation. The first layer convolves 32 8×8 kernels with the input, using a stride of 2 and a zero-padding of 3 (three zeros on each side). The second layer downsamples the output of the first layer with a 2×2 non-overlapping max pooling. The third layer convolves 64 3×3 kernels with a stride of 1 and a zero-padding of size 1 on each border.

4.1 What is the dimensionality of the output of the last layer, i.e. the number of scalars it contains ?

answer :

$$\begin{aligned} w_{\text{out}} &= \left\lfloor \frac{w_{\text{in}} - \text{kemel } + 2 \cdot \text{ padding}}{\text{stride}} \right\rfloor + 1 \\ w_{\text{out}}^{(1)} &= \left\lfloor \frac{128-8+2(3)}{2} \right\rfloor + 1 = [64, 64] \end{aligned}$$

after max-pooling : $w_{\text{out}}^{(2)} = [32, 32]$
output of last layer : $w_{\text{out}}^{(3)} = \left\lfloor \frac{32-3+2}{1} \right\rfloor + 1 = 32$ last layer output $= [32, 32]$

4.2 Not including the biases, how many parameters are needed for the last layer ?

answer :
parameters for the last layer =
filter width * filter height * input feature map * output feature map
= 3 * 3 * 32 * 64 = 18,432

4.3 Compute the *full*, *valid*, and *same* convolution (with kernel flipping) for the following 1D matrices : $[1, 2, 3, 4] * [1, 0, 2]$

answer :
Full = [1, 2, 5, 8, 6, 8]
valid = [5 , 8]
same = [2, 5, 8, 6]

Question 5 (2-5-6-2-5-6)**.** Let us use the notation $*$ and $\tilde{*}$ to denote the valid and full convolution operator **without kernel flipping**, respectively. The operations are defined as

$$\text{valid} : (\boldsymbol{x} * \boldsymbol{w})_n = \sum_{j=1}^{k} x_{n+j-1} w_j \tag{4}$$

$$\text{full} : (\boldsymbol{x} \tilde{*} \boldsymbol{w})_n = \sum_{j=1}^{k} x_{n+j-k} w_j \ , \tag{5}$$

where k is the size of the kernel $\boldsymbol{w}$. As a convention, the value of a vector indexed "out-of-bound" is zero, e.g. if $\boldsymbol{x} \in \mathbb{R}^d$, then $x_i = 0$ for $i < 1$ and $i > d$. We define the flip operator which reverse the ordering of the components of a vector, i.e. $\text{flip}(\boldsymbol{w})_j = w_{k-j+1}$.

Consider a convolutional network with 1-D input $\boldsymbol{x} \in \mathbb{R}^d$. Its first and second convolutional layers have kernel $\boldsymbol{w}^1 \in \mathbb{R}^{k_1}$ and $\boldsymbol{w}^2 \in \mathbb{R}^{k_2}$, respectively. Assume $k_1 < d$ and $k_2 < d$. The network is specified as follows :

$$\begin{aligned}
\boldsymbol{a}^1 &\leftarrow \boldsymbol{x} * \boldsymbol{w}^1 \text{ (valid convolution)} && (6)\\
\boldsymbol{h}^1 &\leftarrow \text{ReLU}(\boldsymbol{a}^1) && (7)\\
\boldsymbol{a}^2 &\leftarrow \boldsymbol{h}^1 * \boldsymbol{w}^2 \text{ (valid convolution)} && (8)\\
\boldsymbol{h}^2 &\leftarrow \text{ReLU}(\boldsymbol{a}^2) && (9)\\
&\ldots && (10)\\
L &\leftarrow \ldots && (11)
\end{aligned}$$

where L is the loss.

5.1 What is the dimensionality of $\boldsymbol{a}^2$? Denote it by $|\boldsymbol{a}^2|$.

answer :

$$\begin{aligned}
w_{\text{out}} &= \left[\frac{w_{in} - kernel + 2 * \text{ padding}}{\text{stride}}\right] + 1\\
\left|a^{(1)}\right| = w_{\text{out}} &= \left[\frac{d-k_1}{1}\right] + 1 = d - k_1 + 1\\
\left|a^{(2)}\right| = w_{\text{out}} &= \left[\frac{d-k_1+1-k_2}{1}\right] + 1 = d - k_1 - k_2 + 2
\end{aligned}$$

5.2 Derive $\frac{\partial a_i^2}{\partial h_n^1}$. Answer for all $i \in \{1, ..., |\boldsymbol{a}^2|\}$, given a particular n.

answer :

$$\left(a^2\right)_i = \left(h^1 * w^2\right)_i = \sum_{j=1}^{k_2} h_{i+j-1}^1 w_j = h_i^1 w_1 + h_{i+1}^1 w_2 + \cdots + h_{i+k_2-1}^1 w_{k_2}$$

$$\frac{\partial a_i^2}{\partial h_n^1} = \begin{cases} w_{n-i+1} & i \leqslant n \leqslant i + k_2 - 1 \\ 0 & 0.w \end{cases} \quad \text{or} \ : \frac{\partial a_i^2}{\partial h_n^1} = \begin{cases} w_{n-i+1} & n - k_2 + 1 \leqslant i \leqslant n \\ 0 & 0.w \end{cases}$$

5.3 Show that $\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L \tilde{*} \operatorname{flip}(\boldsymbol{w}^2)$ (full convolution). Start with

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1}$$

answer :

$$\begin{aligned}
&\left(\nabla_{h^1} L\right)_n = \sum_{i=1}^{\left|a^2\right|} \left(\nabla_{a^2} L\right)_i \frac{\partial a_i^2}{\partial h_n^1} = \sum_{i=n-k_2+1}^{n} \left(\nabla_{a^2} L\right)_i \frac{\partial a_i^2}{\partial h_n^1}\\
&\quad \left(\nabla_{a^2} L\right)_{n-k_2+1} w_{k_2}^2 + \left(\nabla_{a^{2L}}\right)_{n-k_2+2} w_{k^2+1}^2 + \cdots + \left(\nabla_{a^2} L\right)_n w_1^2\\
&= \sum_{j=1}^{k_2} \left(\nabla_{a^2} L\right)_{n-k_2+j} \quad w_{k_2-j+1}^2 = \sum_{j=1}^{k_2} \left(\nabla_{a^2} L\right)_{n-k_2+j} \operatorname{flip}\left(w^2\right)_j\\
&\overset{(5)}{\Rightarrow} = \left(\nabla_{a^2} L \tilde{*} \operatorname{flip}\left(w^2\right)\right)_n
\end{aligned}$$

For the following, assume the convolutions in equations (6) and (8) are **full instead of valid**.

5.4 What is the dimensionality of $\boldsymbol{a}^2$? Denote it by $|\boldsymbol{a}^2|$.

answer :

$$w_{\text{out}} = \left\lfloor \frac{w_{\text{in}} - \text{ kernel } + 2*(\text{ kernel } -1)}{\text{stride}} \right| +1$$
$$|a^2| = \left\lfloor \frac{d-k_1+2*(k_1-1)}{1} \right| +1 = d+k_1-1$$
$$|a^2| = \left\lfloor \frac{d+k_1-1-k_2+2*(k_2-1)}{1} \right| +1 = d+k_1+k_2-2$$

5.5 Derive $\frac{\partial a_i^2}{\partial h_n^1}$. Answer for all $i \in \{1, ..., |\boldsymbol{a}^2|\}$, given a particular n.

answer :

$$(a^2)_i = (h^1 \tilde{*} w^2)_i = \sum_{j=1}^{k_2} h^1_{i+j-k_2} w_j = h^1_{i+1-k_2} w_1 + h^1_{i+1-k_2} w_2 + \cdots + h^1_i w_{k_2}$$

$$\frac{\partial a_i^2}{\partial h_n^1} = \begin{cases} w_{n-i+k_2} & i+1-k_2 \leqslant n \leqslant i \\ 0 & o.w \end{cases}$$

$$\text{or } : \frac{\partial a_i^2}{\partial h_n^1} = \begin{cases} w_{n-i+k_2} & n \leqslant i \leqslant n+k_2-1 \\ 0 & 0.w \end{cases}$$

Show that $\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L * \text{flip}(\boldsymbol{w}^2)$ (valid convolution). Start with

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1}$$

answer :

$$(\nabla_{h^1} L)_n = \sum_{i=1}^{|a^2|} (\nabla_{a^2} L)_i \frac{\partial a_i^2}{\partial h_n^1} = \sum_{i=n}^{n+k_2-1} (\nabla_{a^2} L)_i \frac{\partial a_i^2}{\partial h_n^1}$$
$$(\nabla_{a^2} L)_n w^2_{k_2} + (\nabla_{a^2 L})_{n+1} w^2_{k^2-1} + \cdots + (\nabla_{a^2} L)_{n+k_2-1} w^2_1$$
$$= \sum_{j=1}^{k_2} (\nabla_{a^2} L)_{n-1+j} \quad w^2_{k_2-j+1} = \sum_{j=1}^{k_2} (\nabla_{a^2} L)_{n-1+j} \text{flip}\left(w^2\right)_j$$
$$\overset{(4)}{\Rightarrow} = (\nabla_{a^2} L * \text{flip}(w^2))_n$$