Homework 3 – Deep Learning (CS/DS 541, Murai, Fall 2023)

You may complete this homework assignment either individually or in teams up to 2 people.

1. **Derivation of softmax regression gradient updates** [20 points]: As explained in class, let

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}^{(1)} & \dots & \mathbf{w}^{(c)} \end{bmatrix}$$

be an $m \times c$ matrix containing the weight vectors from the c different classes. The output of the softmax regression neural network is a vector with c dimensions such that:

$$\begin{aligned} \hat{y}_k &= \frac{\exp z_k}{\sum_{k'=1}^{c} \exp z_{k'}} \\ z_k &= \mathbf{x}^\top \mathbf{w}^{(k)} + b_k \end{aligned} \tag{1}$$

for each $k = 1, \dots, c$. Correspondingly, our cost function will sum over all c classes:

$$f_{\text{CE}}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \log \hat{y}_k^{(i)}$$

Important note: When deriving the gradient expression for each weight vector $\mathbf{w}^{(l)}$, it is crucial to keep in mind that the weight vector for each class $l \in \{1, \dots, c\}$ affects the outputs of the network for *every* class, *not* just for class l. This is due to the normalization in Equation 1 – if changing the weight vector *increases* the value of $\hat{y}_l$, then it necessarily must *decrease* the values of the other $\hat{y}_{l' \neq l}$.

In this homework problem, please complete the following derivation that is outlined below:

Derivation: For each weight vector $\mathbf{w}^{(l)}$, we can derive the gradient expression as:

$$\begin{aligned} \nabla_{\mathbf{w}^{(l)}} f_{\text{CE}}(\mathbf{W}, \mathbf{b}) &= -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \nabla_{\mathbf{w}^{(l)}} \log \hat{y}_k^{(i)} \\ &= -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \left(\frac{\nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)}}{\hat{y}_k^{(i)}} \right) \end{aligned}$$

We handle the two cases $l = k$ and $l \neq k$ separately. For $l = k$:

$$\begin{aligned} \nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)} &= \text{complete me...} \\ &= \mathbf{x}^{(i)} \hat{y}_l^{(i)} (1 - \hat{y}_l^{(i)}) \end{aligned}$$

For $l \neq k$:

$$\begin{aligned} \nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)} &= \text{complete me...} \\ &= -\mathbf{x}^{(i)} \hat{y}_k^{(i)} \hat{y}_l^{(i)} \end{aligned}$$

To compute the total gradient of f_{CE} w.r.t. each $\mathbf{w}^{(k)}$, we have to sum over all examples *and* over $l = 1, \dots, c$. (**Hint**: $\sum_k a_k = a_l + \sum_{k \neq l} a_k$. Also, $\sum_k y_k = 1$.)

$$\begin{aligned} \nabla_{\mathbf{w}^{(l)}} f_{\text{CE}}(\mathbf{W}, \mathbf{b}) &= -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \nabla_{\mathbf{w}^{(l)}} \log \hat{y}_k^{(i)} \\ &= \text{complete me...} \\ &= -\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(y_l^{(i)} - \hat{y}_l^{(i)} \right) \end{aligned}$$

Finally, show that

$$\nabla_{\mathbf{b}} f_{\mathrm{CE}}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)} \right)$$

Answer: For $l = k$:

$$\begin{aligned}
\nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)} &= \nabla_{\mathbf{w}^{(l)}} \left[\frac{\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(l)}}{\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')}} \right] \\
&= \mathbf{x}^{(i)} \frac{\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(l)}}{\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')}} - \mathbf{x}^{(i)} \frac{\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(l)}}{\left(\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')} \right)^2} \exp(\mathbf{x}^{(i)\top} \mathbf{w}^{(l)}) \\
&= \mathbf{x}^{(i)} \left[\frac{\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(l)}}{\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')}} - \frac{(\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(l)})^2}{\left(\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')} \right)^2} \right] \\
&= \mathbf{x}^{(i)} \left[\hat{y}_l^{(i)} - \left(\hat{y}_l^{(i)} \right)^2 \right] \\
&= \mathbf{x}^{(i)} \hat{y}_l^{(i)} (1 - \hat{y}_l^{(i)})
\end{aligned}$$

For $l \neq k$:

$$\begin{aligned}
\nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)} &= \nabla_{\mathbf{w}^{(l)}} \left[\frac{\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k)}}{\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')}} \right] \\
&= -\mathbf{x}^{(i)} \frac{\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k)}}{(\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')})^2} \exp \mathbf{x}^{(i)\top} \mathbf{w}_l \\
&= -\mathbf{x}^{(i)} \frac{(\exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k)})(\exp \mathbf{x}^{(i)\top} \mathbf{w}_l)}{\left(\sum_{k'=1}^{c} \exp \mathbf{x}^{(i)\top} \mathbf{w}^{(k')} \right)^2} \\
&= -\mathbf{x}^{(i)} \hat{y}_k^{(i)} \hat{y}_l^{(i)}
\end{aligned}$$

To compute the total gradient of f_{CE} w.r.t. each $\mathbf{w}^{(k)}$, we have to sum over all examples *and* over

$l = 1, \ldots, c$:

$$
\begin{aligned}
\nabla_{\mathbf{w}^{(l)}} f_{\text{CE}}(\mathbf{W}, \mathbf{b}) &= -\sum_{i=1}^{m} \sum_{k=1}^{c} y_k^{(i)} \nabla_{\mathbf{w}^{(l)}} \log \hat{y}_k^{(i)} \\
&= -\sum_{i=1}^{m} \mathbf{x}^{(i)} \left[\frac{y_l^{(i)} \hat{y}_l^{(i)} (1 - \hat{y}_l^{(i)})}{\hat{y}_l^{(i)}} - \sum_{k \neq l} \frac{y_k^{(i)} \hat{y}_k^{(i)} \hat{y}_l^{(i)}}{\hat{y}_k^{(i)}} \right] \\
&= -\sum_{i=1}^{m} \mathbf{x}^{(i)} \left[y_l^{(i)} (1 - \hat{y}_l^{(i)}) - \sum_{k \neq l} y_k^{(i)} \hat{y}_l^{(i)} \right] \\
&= -\sum_{i=1}^{m} \mathbf{x}^{(i)} \left[y_l^{(i)} (1 - \hat{y}_l^{(i)}) + y_l^{(i)} \hat{y}_l^{(i)} - \sum_{k} y_k^{(i)} \hat{y}_l^{(i)} \right] \\
&= -\sum_{i=1}^{m} \mathbf{x}^{(i)} \left[y_l^{(i)} - y_l^{(i)} \hat{y}_l^{(i)} + y_l^{(i)} \hat{y}_l^{(i)} - \hat{y}_l^{(i)} \sum_{k} y_k^{(i)} \right] \\
&= -\sum_{i=1}^{m} \mathbf{x}^{(i)} \left[y_l^{(i)} - \hat{y}_l^{(i)} \right]
\end{aligned}
$$

The gradient w.r.t. $\mathbf{b}$ is derived exactly the same way as for $\nabla_{\mathbf{W}}$, except that there is no $\mathbf{x}^{(i)}$ term. Hence, we have

$$
\nabla_{b_l} f_{\text{CE}}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_l^{(i)} - \hat{y}_l^{(i)} \right] \tag{2}
$$

Combining all these (scalar) gradients into one vector, we obtain:

$$
\nabla \mathbf{b} f_{\text{CE}}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \left[\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)} \right]
$$

2. **Derivation of Cross-Entropy as Negative Log-Likelihood** [10 points]: *Answer*:

$$
\begin{aligned}
\text{NLL} &= -\log P(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n)} \mid \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}, \mathbf{W}, \mathbf{b}) && (3) \\
&= -\log \prod_{i=1}^{n} P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{W}, \mathbf{b}) && (4) \\
&= -\log \prod_{i=1}^{n} \prod_{k=1}^{c} \left(\hat{y}_k^{(i)} \right)^{\left(y_k{}^{(i)} \right)} && (5) \\
&= -\sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \log \hat{y}_k^{(i)} && (6) \\
&= f_{\text{CE}} && (7)
\end{aligned}
$$

3. **Implementation of softmax regression** [20 points]:

Train a 2-layer softmax neural network to classify images of fashion items (10 different classes, such as shoes, t-shirts, dresses, etc.) from the Fashion MNIST dataset. The input to the network will be a 28×28-pixel image (converted into a 784-dimensional vector); the output will be a vector of 10 probabilities (one for each class). The cross-entropy loss function that you minimize should be

$$f_{\text{CE}}(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(10)}, b^{(1)}, \ldots, b^{(10)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10} y_k^{(i)} \log \hat{y}_k^{(i)} + \frac{\alpha}{2}\sum_{k=1}^{c} \mathbf{w}^{(k)\top}\mathbf{w}^{(k)}$$

where n is the number of examples and α is a regularization constant.. Note that each $\hat{y}_k$ implicitly depends on all the weights $\mathbf{W} = \left[\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(10)}\right]$ and biases $\mathbf{b} = \left[b^{(1)}, \ldots, b^{(10)}\right]$.

To get started, first download the Fashion MNIST dataset from the following web links:

- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_images.npy`
- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_labels.npy`
- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_images.npy`
- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_labels.npy`

These files can be loaded into `numpy` using `np.load`. Each "labels" file consists of a 1-d array containing n labels (valued 0-9), and each "images" file contains a 2-d array of size $n \times 784$, where n is the number of images.

Next, implement stochastic gradient descent (SGD) to minimize the cross-entropy loss function on this dataset. Regularize the weights but *not* the biases. Optimize the same hyperparameters as in homework 2 problem 2 (age regression). You should also use the same methodology as for the previous homework, including the splitting of the training files into validation and training portions.

Performance evaluation: Once you have tuned the hyperparameters and optimized the weights so as to maximize performance on the validation set, then: (1) **stop** training the network and (2) evaluate the network on the **test** set. Record the performance both in terms of (unregularized) cross-entropy loss (smaller is better) and percent correctly classified examples (larger is better); put this information into the PDF you submit.

Answer: Here are the key methods for performing the softmax, computing cross-entropy, and computing its gradient:

```
def softmax (z):
    denom = np.sum(np.exp(z), axis=1, keepdims=True)
    return np.exp(z) / denom

def fCE (W, X, Y, alpha = 0.):
    Yhat = softmax(X.T.dot(W))
    cost = - 1./X.shape[1] * np.sum(Y * np.log(Yhat))
    return cost

def gradK (W, X, Y, alpha = 0.):
    Yhat = softmax(X.T.dot(W))
    reg = alpha * W
    dfCEdw = 1./X.shape[1] * X.dot(Yhat - Y) + reg
    dfCEdb = 1./X.shape[1] * np.sum(Yhat - Y, axis=0)
    return dfCEdw, dfCEdb

def computeAccuracy (W, X, Y):
    Yhat = softmax(X.T.dot(W))
    return np.mean(np.argmax(Y, axis=1) == np.argmax(Yhat, axis=1))
```

4. **Implementing gradient computation via the multivariate chain rule (linear case)** [20 points]: Consider the two vector fields below taken from Class 5, slide 43:

$$f\left(\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}\right) = [x_1 - 2x_2 + x_3/4] \quad g\left(\begin{bmatrix}x_1\\x_2\end{bmatrix}\right) = \begin{bmatrix}x_1+2x_2\\x_2\\-x_1+3\end{bmatrix}.$$

(a) Show how to represent each function as an affine transformation of the form $\mathbf{W}\mathbf{x}+\mathbf{b}$. For function g, denote the parameters by $\mathbf{W}_1$ and $\mathbf{b}_1$, whereas for function f, denote the parameters by $\mathbf{W}_2$ and b_2. [1 point]
Answer:

$$\mathbf{W}_1 = \begin{bmatrix}1 & 2\\0 & 1\\-1 & 0\end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix}0\\0\\3\end{bmatrix} \qquad \mathbf{W}_2 = \begin{bmatrix}1 & -2 & 1/4\end{bmatrix}, \quad b_2 = [0].$$

(b) Implement a python function `AffineTransformation(W, b, x)` that can be used to compute either function by taking as input two arrays of parameters $\mathbf{W}$ and $\mathbf{b}$ and a vector $\mathbf{x}$. [1 point]
Answer:

```
def AffineTransformation(W, b, x):
    return W@x+b
```

(c) Building on the previous function, implement a new function `Composition(L, x)` that calculates the composition of affine transformations represented as a list of pairs $L = [(\mathbf{W}_1, \mathbf{b}_1), \ldots, (\mathbf{W}_n, \mathbf{b}_n)]$ and an initial vector x. **Use the following convention:** the functions are applied from the smallest to the largest index. **Return all the activation values (i.e., intermediate results) as a list variable z** $= (\mathbf{z}_1, \ldots, z_n)$. [2 points]
Answer:

```
def Composition(L, x):
    z = np.empty(len(L))
    for ix, params in enumerate(L):
        W, b = params
        z[ix] = AffineTransformation(W, b, x)
     return z
```

(d) How would you call the previous function to compute $(f \circ g)(\mathbf{x})$? Return ONLY the final output. [1 point]
Answer: `Composition([(W_1,b_1),(W_2,b_2)], x)[-1]`

(e) Given an affine transformation $\mathbf{z}_2 = f(\mathbf{z}_1)$ parameterized by $\mathbf{W}_2$ and $\mathbf{b}_2$, derive $\frac{\partial \mathbf{z}_2}{\partial \mathbf{b}_2}(\mathbf{z}_1)$.[1 point]
Answer:

$$\frac{\partial \mathbf{z}_2}{\partial \mathbf{b}_2}(\mathbf{z}_1) = \mathbf{1}_{1\times 3}.$$

(It is fine to answer more generically using variables to denote the length of $\mathbf{z}_2$ and $\mathbf{b}_2$.)

(f) Given an affine transformation $\mathbf{z} = h(\mathbf{x})$ parameterized by $\mathbf{W}_{n\times m}$ and $\mathbf{b}_{n\times 1}$, we know that

$$\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]}(\mathbf{x}) = \begin{bmatrix}\mathbf{x}^\top & \ldots & \mathbf{0}^\top\\ \vdots & \ddots & \vdots\\ \mathbf{0}^\top & \ldots & \mathbf{x}^\top\end{bmatrix}_{n\times nm}.$$

Also, note that

$$\text{unvec}[\mathbf{A}\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]}(\mathbf{x})] = \mathbf{x}\mathbf{A}.$$

Using the above, write equations for: [2 points]

$$\frac{\partial(f\circ g)}{\partial \mathbf{b}_2}(\mathbf{x}) \quad \text{and} \quad \text{unvec}\left[\frac{\partial(f\circ g)}{\partial \text{vec}[\mathbf{W}_2]}(\mathbf{x})\right]$$

Answer:

$$\frac{\partial(f\circ g)}{\partial \mathbf{b}_2}(\mathbf{x}) = \frac{\partial \mathbf{z}_2}{\partial \mathbf{b}_2}(\mathbf{x}) = \mathbf{I}$$

$$\frac{\partial(f\circ g)}{\partial \text{vec}[\mathbf{W}_2]}(\mathbf{x}) = \frac{\partial \mathbf{z}_2}{\partial \text{vec}[\mathbf{W}_2]}(g(\mathbf{x})) = [\mathbf{z}_1^\top]_{1\times 3}.$$

Since $\mathbf{W}_2$ is 1×3, then the unvectorized Jacobian is just the transpose of the vectorized one:

$$\text{unvec}\left[\frac{\partial(f\circ g)}{\partial \text{vec}[\mathbf{W}_2]}(\mathbf{x})\right] = \mathbf{z}_1.$$

(g) Using the multivariate chain rule, write equations for: [2 points]

$$\frac{\partial(f\circ g)}{\partial \mathbf{b}_1}(\mathbf{x}) \quad \text{and} \quad \text{unvec}\left[\frac{\partial(f\circ g)}{\partial \text{vec}[\mathbf{W}_1]}(\mathbf{x})\right]$$

Answer:

$$\begin{aligned}\frac{\partial(f\circ g)}{\partial \mathbf{b}_1}(\mathbf{x}) &= \frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}\frac{\partial \mathbf{z}_1}{\partial \mathbf{b}_1}(\mathbf{x}) \\ &= \mathbf{W}_2\mathbf{I} \\ &= \mathbf{W}_2\end{aligned}$$

$$\begin{aligned}\frac{\partial(f\circ g)}{\partial \text{vec}[\mathbf{W}_1]}(\mathbf{x}) &= \frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}\frac{\partial \mathbf{z}_1}{\partial \text{vec}[\mathbf{W}_1]}(\mathbf{x}) \\ &= \mathbf{W}_2\begin{bmatrix}\mathbf{x}^\top & \mathbf{0}^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & \mathbf{x}^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & \mathbf{0}^\top & \mathbf{x}^\top\end{bmatrix}_{3\times 6} \\ &= \begin{bmatrix}[\mathbf{W}_2]_1\mathbf{x}^\top & [\mathbf{W}_2]_2\mathbf{x}^\top & [\mathbf{W}_2]_3\mathbf{x}^\top\end{bmatrix}_{1\times 6}\end{aligned}$$

Since $\mathbf{W}_1$ is 3×2, then the unvectorized Jacobian (i.e., the gradient) is a 3×2 matrix:

$$\text{unvec}\left[\frac{\partial(f\circ g)}{\partial \text{unvec}[\mathbf{W}_1]}(\mathbf{x})\right] = \mathbf{x}\mathbf{W}_2$$

Note: in the homework, I defined the unvec operation in a way that gives as a "Jacobian shape", i.e., #outputs x #params (in contrast, in the lecture slides, the unvec gives us the gradient directly, which is the transpose version). Either definition will be accepted.

(h) Using the list of parameters L, the input vector $\mathbf{x}$ and all the activation values `z` obtained from the `Composition`, implement a function `ComputeJacobians(L,x,z)` that returns the Jacobians of z_n (a scalar) for all the parameters in L, i.e., $\partial z_n/\partial[\mathbf{b}_i]$. and $\text{unvec}[\partial z_n/\partial \text{vec}[\mathbf{W}_i]]$. **To earn full marks, the solution should apply to any $n \geq 2$.** [10 points]

```
def ComputeJacobians(L,x,z):
    n = len(L)
    W, b = zip(*L)
    db = [[]]*n
```

```
dW = [[]]*n

db[n-1] = np.array([[1]]) // add jacobian for b_n
dW[n-1] = db[n-1] @ z[n-2] // add jacobian for W_n

for ix in range(n-2,0,-1): // compute jacobians
    db[ix] = db[ix+1] @ W[ix+1]
    dW[ix] = z[ix-1] @ db[ix]

db[0]  = db[1] @ W[1]
dW[0] = x @ db[0]
```

Put your code in a Python file called `homework3_WPIUSERNAME1.py` (or `homework3_WPIUSERNAME1_WPIUSERNAME3.py` for teams). For the proof and derivation, as well as the cross-entropy values from the Fashion MNIST problem, please create a PDF called `homework3_WPIUSERNAME1.pdf` (or `homework3_WPIUSERNAME1_WPIUSERNAME3.pdf` for teams). Create a Zip file containing both your Python and PDF files, and then submit on Canvas.