Homework 1: Backpropagation

DS-GA 1008 Deep Learning

Spring 2024

The goal of homework 1 is to help you understand the common techniques used in Deep Learning and how to update network parameters by the using backpropagation algorithm.

Part 1 has two sub-parts, 1.1, 1.2, 1.3 majorly deal with the theory of backpropagation algorithm whereas 1.4 is to test conceptual knowledge on deep learning. For part 1.2 and 1.3, you need to answer the questions with mathematical equations. You should put all your answers in a PDF file and we will not accept any scanned hand-written answers. It is recommended to use LaTeX.

For part 2, you need to program in Python. It requires you to implement your own forward and backward pass without using autograd. You need to submit your `mlp.py` file for this part.

Submit the following files in a zip file `your_net_id.zip` through NYU Brightspace:

- `theory.pdf`
- `mlp.py`
- `sgd.py`

The following behaviors will result in penalty of your final score:

1. 5% penalty for submitting your files without using the correct format. (including naming the zip file, PDF file or python file wrong, or adding extra files in the zip folder, like the testing scripts from part 2).
2. 20% penalty for code submission that cannot be executed using the steps we mentioned in part 2. So please test your code before submit it.

1 Theory (50pt)

To answer questions in this part, you need some basic knowledge of linear algebra and matrix calculus. Also, you need to follow the instructions:

1. Every provided vector is treated as column vector.
2. IMPORTANT: You need to use the numerator-layout notation for matrix calculus. Please refer to Wikipedia about the notation. Specifically, $\frac{\partial y}{\partial \boldsymbol{x}}$ is a row-vector whereas $\frac{\partial \boldsymbol{y}}{\partial x}$ is a column-vector.
3. You are only allowed to use vector and matrix. You cannot use tensor in any of your answer.
4. Missing transpose are considered as wrong answer.

1.1 Two-Layer Neural Nets

You are given the following neural net architecture:

$$\texttt{Linear}_1 \rightarrow f \rightarrow \texttt{Linear}_2 \rightarrow g$$

where $\texttt{Linear}_i(x) = \boldsymbol{W}^{(i)}\boldsymbol{x} + \boldsymbol{b}^{(i)}$ is the i-th affine transformation, and f, g are element-wise nonlinear activation functions. When an input $\boldsymbol{x} \in \mathbb{R}^n$ is fed to the network, $\tilde{\boldsymbol{y}} \in \mathbb{R}^K$ is obtained as the output.

1.1.1 Regression Task

We would like to perform regression task. We choose $f(\cdot) = 3(\cdot)^+ = 3\texttt{ReLU}(\cdot)$ and g to be the identity function. To train this network, we want to minimize the energy loss L and this is computed via the squared Euclidean distance cost C, such that $L(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}) = F(\boldsymbol{x}, \boldsymbol{y}) = C(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \|\tilde{\boldsymbol{y}} - \boldsymbol{y}\|^2$, where $\boldsymbol{y}$ is the output target.

(a) (1pt) Name and mathematically describe the 5 programming steps you would take to train this model with `PyTorch` using SGD on a single batch of data.

(1) forward pass (2) loss computation (3) clear gradients (4) compute gradients (backward pass) (5) update parameters (step)

(b) (4pt) For a single data point (x, y), write down all inputs and outputs for forward pass of each layer. You can only use variable $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}$ in your answer. (note that $\texttt{Linear}_i(\boldsymbol{x}) = \boldsymbol{W}^{(i)}\boldsymbol{x} + \boldsymbol{b}^{(i)}$).

Layer	Input	Output
$\texttt{Linear}_1$	x	$W^{(1)}x+b^{(1)}$
f	$W^{(1)}x+b^{(1)}$	$5(W^{(1)}x+b^{(1)})^+$
$\texttt{Linear}_2$	$5(Wx+b)^+$	$5W^{(2)}(W^{(1)}x+b^{(1)})^++b^{(2)}$
g	$5W^{(2)}(W^{(1)}x+b^{(1)})^++b^{(2)}$	$5W^{(2)}(W^{(1)}x+b^{(1)})^++b^{(2)}$
$\texttt{Loss}$	$5W^{(2)}(W^{(1)}x+b^{(1)})^++b^{(2)}, y$	$\|5W^{(2)}(W^{(1)}x+b^{(1)})^++b^{(2)}-y\|^2$

(c) (6pt) Write down the gradients calculated from the backward pass. You can only use the following variables: $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}, \frac{\partial C}{\partial \tilde{\boldsymbol{y}}}, \frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{s}_1}, \frac{\partial \tilde{\boldsymbol{y}}}{\partial \boldsymbol{s}_2}$ in your answer, where $\boldsymbol{s}_1, \boldsymbol{a}_1, \boldsymbol{s}_2, \tilde{\boldsymbol{y}}$ are the outputs of $\texttt{Linear}_1, f, \texttt{Linear}_2, g$.

First, let's figure out the dimension of each vector, matrix, tensor:

We already have $x \in \mathbb{R}^n$ and $z_3, \hat{y} \in \mathbb{R}^K$, so let's assume $z_1, z_2 \in \mathbb{R}^a$, then we have

$$W^{(1)} \in \mathbb{R}^{a\times n}, \quad b^{(1)} \in \mathbb{R}^a$$
$$W^{(2)} \in \mathbb{R}^{K\times a}, \quad b^{(2)} \in \mathbb{R}^K$$

Using the numerator-layout notation for the matrix calculus we have:

The dimension of all the gradients are

$$\frac{\partial \ell}{\partial W^{(1)}} \in \mathbb{R}^{n\times a}, \quad \frac{\partial \ell}{\partial b^{(1)}} \in \mathbb{R}^{1\times a}$$
$$\frac{\partial \ell}{\partial W^{(2)}} \in \mathbb{R}^{a\times K}, \quad \frac{\partial \ell}{\partial b^{(2)}} \in \mathbb{R}^{1\times K}$$

and

$$\frac{\partial \ell}{\partial \hat{y}} \in \mathbb{R}^{1\times K}, \quad \frac{\partial \hat{y}}{\partial z_3} \in \mathbb{R}^{K\times K}, \quad \frac{\partial z_3}{\partial z_2} \in \mathbb{R}^{K\times a}, \quad \frac{\partial z_2}{\partial z_1} \in \mathbb{R}^{a\times a}$$
$$\frac{\partial z_1}{\partial b^{(1)}} \in \mathbb{R}^{a\times a}, \quad \frac{\partial z_1}{\partial W^{(1)}} \in \mathbb{R}^{a\times n\times a}$$
$$\frac{\partial z_3}{\partial b^{(2)}} \in \mathbb{R}^{K\times K}, \quad \frac{\partial z_3}{\partial W^{(2)}} \in \mathbb{R}^{K\times a\times K}$$

It is easy to see that

$$\frac{\partial \ell}{\partial z_3} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3}$$
$$\frac{\partial \ell}{\partial z_1} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial z_1} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} W^{(2)} \frac{\partial z_2}{\partial z_1}$$

For bias, we have

$$\frac{\partial \ell}{\partial b^{(1)}} = \frac{\partial \ell}{\partial z_1}\frac{\partial z_1}{\partial b^{(1)}} = \frac{\partial \ell}{\partial z_1}$$
$$\frac{\partial \ell}{\partial b^{(2)}} = \frac{\partial \ell}{\partial z_3}\frac{\partial z_3}{\partial b^{(2)}} = \frac{\partial \ell}{\partial z_3}$$

For weight, since $\frac{\partial z_1}{\partial W^{(1)}}$ and $\frac{\partial z_3}{\partial W^{(2)}}$ are tensors, so we need to do an extra step:

$$\frac{\partial \ell}{\partial z_1} = \left[\frac{\partial \ell}{\partial (z_1)_1}\frac{\partial \ell}{\partial (z_1)_2}\cdots\frac{\partial \ell}{\partial (z_1)_a}\right]$$
$$\frac{\partial z_1}{\partial W^{(1)}} = \begin{bmatrix} \frac{\partial (z_1)_1}{\partial W^{(1)}} \\ \frac{\partial (z_1)_2}{\partial W^{(1)}} \\ \vdots \\ \frac{\partial (z_1)_a}{\partial W^{(1)}} \end{bmatrix}$$
$$\frac{\partial \ell}{\partial z_3} = \left[\frac{\partial \ell}{\partial (z_3)_1}\frac{\partial \ell}{\partial (z_3)_2}\cdots\frac{\partial \ell}{\partial (z_1)_K}\right]$$
$$\frac{\partial z_3}{\partial W^{(2)}} = \begin{bmatrix} \frac{\partial (z_3)_1}{\partial W^{(2)}} \\ \frac{\partial (z_3)_2}{\partial W^{(2)}} \\ \vdots \\ \frac{\partial (z_3)_K}{\partial W^{(2)}} \end{bmatrix}$$

Then we have

$$\frac{\partial \ell}{\partial W^{(1)}} = \frac{\partial \ell}{\partial z_1}\frac{\partial z_1}{\partial W^{(1)}} = \sum_i \frac{\partial \ell}{\partial (z_1)_i}\frac{\partial (z_1)_i}{\partial W^{(1)}} = x\frac{\partial \ell}{\partial z_1}$$
$$\frac{\partial \ell}{\partial W^{(2)}} = \frac{\partial \ell}{\partial z_3}\frac{\partial z_3}{\partial W^{(2)}} = \sum_i \frac{\partial \ell}{\partial (z_3)_i}\frac{\partial (z_3)_i}{\partial W^{(2)}} = z_2\frac{\partial \ell}{\partial z_3}$$

So the answer is

Parameter	Gradient
$W^{(1)}$	$x\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}W^{(2)}\frac{\partial z_2}{\partial z_1}$
$b^{(1)}$	$\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}W^{(2)}\frac{\partial z_2}{\partial z_1}$
$W^{(2)}$	$(W^{(1)}x + b^{(1)})^+\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}$
$b^{(2)}$	$\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}$

or

Parameter	Gradient
$W^{(1)}$	$(x\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}W^{(2)}\frac{\partial z_2}{\partial z_1})^T$
$b^{(1)}$	$(\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}W^{(2)}\frac{\partial z_2}{\partial z_1})^T$
$W^{(2)}$	$((W^{(1)}x+b^{(1)})^+\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3})^T$
$b^{(2)}$	$(\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3})^T$

(d) (2pt) Show us the elements of $\frac{\partial \boldsymbol{a_1}}{\partial \boldsymbol{s_1}}$, $\frac{\partial \tilde{\boldsymbol{y}}}{\partial \boldsymbol{s_2}}$ and $\frac{\partial C}{\partial \tilde{\boldsymbol{y}}}$ (be careful about the dimensionality)?

$\frac{\partial a_1}{\partial s_1}$ and $\frac{\partial \tilde{y}}{\partial s_2}$ are matrices and $\frac{\partial \ell}{\partial \tilde{y}}$ is row vector.

All of the off diagonal elements of $\frac{\partial a_1}{\partial s_1}$ and $\frac{\partial \tilde{y}}{\partial s_2}$ are 0.

For $\frac{\partial a_1}{\partial s_1}$, the diagonal elements are

$$\left(\frac{\partial a_1}{\partial s_1}\right)_{ii} = 3\mathbf{1}_{[(s_1)_i>0]}$$

For $\frac{\partial \tilde{y}}{\partial s_2}$, the diagonal elements are

$$\left(\frac{\partial \tilde{y}}{\partial s_2}\right)_{ii} = 1$$

For $\frac{\partial \tilde{y}}{\partial s_2}$, the elements are

$$\left(\frac{\partial \ell}{\partial \tilde{y}}\right)_{i} = 2(\tilde{y}-y)_i$$

1.1.2 Classification Task

We would like to perform multi-class classification task, so we set $f = \tanh$ and $g = \sigma$, the logistic sigmoid function $\sigma(x) \doteq (1+\exp(-x))^{-1}$.

(a) (2pt + 3pt + 1pt) If you want to train this network, what do you need to change in the equations of (b), (c) and (d), assuming we are using the same squared Euclidean distance loss function.

Layer	Input	Output
$\texttt{Linear}_1$	x	$W^{(1)}x+b^{(1)}$
f	$W^{(1)}x+b^{(1)}$	$\tanh(W^{(1)}x+b^{(1)})$
$\texttt{Linear}_2$	$\tanh(W^{(1)}x+b^{(1)})$	$W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}$
g	$W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}$	$\sigma(W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)})$
$\texttt{Loss}$	$\sigma(W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}), y$	$\|\sigma(W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)})-y\|^2$

Parameter	Gradient
$W^{(1)}$	$x\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}W^{(2)}\frac{\partial a_1}{\partial s_1}$
$b^{(1)}$	$\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}W^{(2)}\frac{\partial a_1}{\partial s_1}$
$W^{(2)}$	$\tanh(W^{(1)}x+b^{(1)})\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}$
$b^{(2)}$	$\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}$

All of the off diagonal elements of $\frac{\partial a_1}{\partial s_1}$ and $\frac{\partial \tilde{y}}{\partial s_2}$ are 0.

$$\left(\frac{\partial a_1}{\partial s_1}\right)_{ii} = (1-\tanh((s_1)_i^2))$$

$$\left(\frac{\partial \tilde{y}}{\partial s_2}\right)_{ii} = \sigma((s_2)_i)(1-\sigma((s_2)_i))$$

$$\left(\frac{\partial \ell}{\partial \tilde{y}}\right)_i = 2(\tilde{y}-y)_i$$

(b) (2pt + 3pt + 1pt) Now you think you can do a better job by using a *Binary Cross Entropy* (BCE) loss function $D_{\text{BCE}}(\boldsymbol{y},\tilde{\boldsymbol{y}}) = \frac{1}{K}\sum_{i=1}^{K} -\left[y_i \log(\tilde{y}_i) + (1-y_i)\log(1-\tilde{y}_i)\right]$. What do you need to change in the equations of (b), (c) and (d)?

Layer	Input	Output
$\texttt{Linear}_1$	x	$W^{(1)}x+b^{(1)}$
f	$W^{(1)}x+b^{(1)}$	$\tanh(W^{(1)}x+b^{(1)})$
$\texttt{Linear}_2$	$\tanh(W^{(1)}x+b^{(1)})$	$W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}$
g	$W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}$	$\sigma(W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)})$
$\texttt{Loss}$	$\sigma(W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}), y$	$-\frac{1}{K}\left[y^T \log(\sigma(W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}))\right.$ $\left.+(1-y)^T \log(1-\sigma(W^{(2)}\tanh(W^{(1)}x+b^{(1)})+b^{(2)}))\right]$

Parameter	Gradient
$W^{(1)}$	$x\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}W^{(2)}\frac{\partial a_1}{\partial s_1}$
$b^{(1)}$	$\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}W^{(2)}\frac{\partial a_1}{\partial s_1}$
$W^{(2)}$	$\tanh(W^{(1)}x+b^{(1)})\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}$
$b^{(2)}$	$\frac{\partial \ell}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial s_2}$

All of the off diagonal elements of $\frac{\partial a_1}{\partial s_1}$ and $\frac{\partial \tilde{y}}{\partial s_2}$ are 0.

$$\left(\frac{\partial a_1}{\partial s_1}\right)_{ii} = (1 - \tanh((s_1)_i^2))$$

$$\left(\frac{\partial \tilde{y}}{\partial s_2}\right)_{ii} = \sigma((s_2)_i)(1 - \sigma((s_2)_i))$$

$$\left(\frac{\partial \ell}{\partial \tilde{y}}\right)_i = \frac{1}{K}\frac{y_i - \tilde{y}_i}{\tilde{y}_i - \tilde{y}_i^2}$$

(c) (1pt) Things are getting better. You realize that not all intermediate hidden activations need to be binary (or soft version of binary). You decide to use $f(\cdot) = (\cdot)^+$ but keep g as σ. Explain why this choice of f can be beneficial for training a (deeper) network.

no vanishing gradient

1.2 Conceptual Questions

(a) (1pt) Why is softmax actually softargmax?

It gives a soft version of argmax. It gives the argmax smooth gradients

(b) (3pt) Draw the computational graph defined by this function, with inputs $x, y, z \in \mathbb{R}$ and output $w \in \mathbb{R}$. You make use symbols x, y, z, o, and operators $*, +$ in your solution. Be sure to use the correct shape for symbols and operators as shown in class.

$$a = xy + z$$
$$b = a(x + x)$$
$$w = ab + b$$

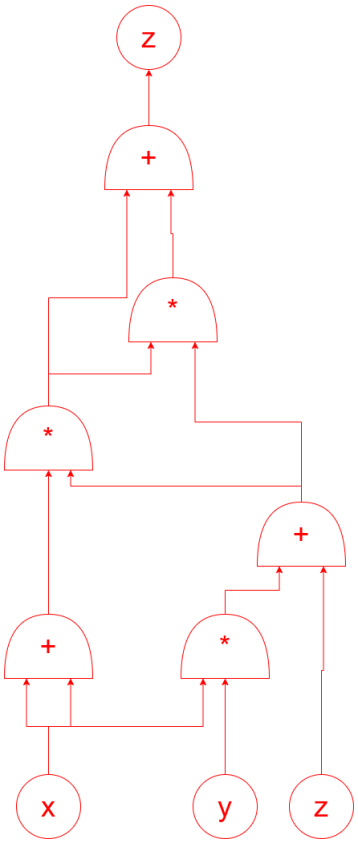

Figure 1: Computation Graph

(c) (2pt) Draw the graph of the derivative for the following functions?

- `GELU()`
- `LeakyReLU(negative_slope=0.1)`
- `ReLU()`
- `Tanh()`

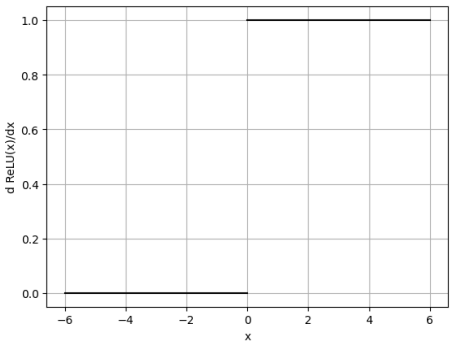

Figure 2: Derivative of `ReLU()`

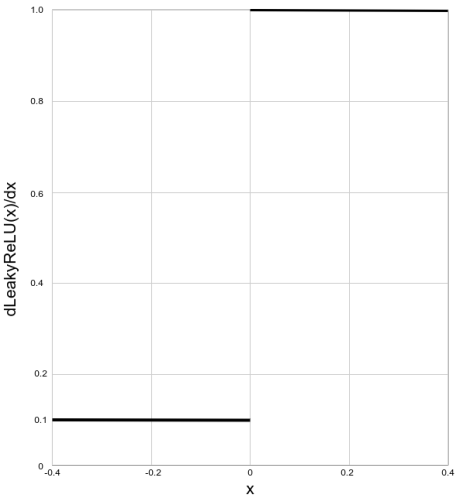

Figure 3: Derivative of `LeakyReLU()`

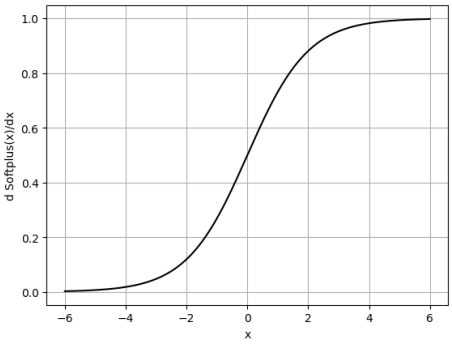

Figure 4: Derivative of `Softplus()`

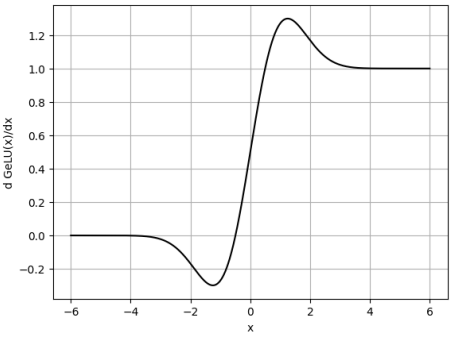

Figure 5: Derivative of `GeLU()`

(d) (3pt) Given function $f(\boldsymbol{x}) = \boldsymbol{W}_1\boldsymbol{x}$ with $\boldsymbol{W}_1 \in \mathbb{R}^{b\times a}$ and $g(\boldsymbol{x}) = \boldsymbol{W}_2\boldsymbol{x}$ with $W_2 \in \mathbb{R}^{b\times a}$:

(a) What is the Jacobian matrix of f and g

- Jacobian of f is W_1

- Jacobian of g is W_2

(b) What is the Jacobian matrix of $h(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$

Jacobian of h is $W_1 + W_2$

(c) What is the Jacobian matrix of $h(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$ if $\boldsymbol{W}_1 = \boldsymbol{W}_2$

Jacobian of h is $2 \cdot W_1$

(e) (3pt) Given function $f(\boldsymbol{x}) = \boldsymbol{W}_1\boldsymbol{x}$ with $\boldsymbol{W}_1 \in \mathbb{R}^{b\times a}$ and $g(\boldsymbol{x}) = \boldsymbol{W}_2\boldsymbol{x}$ with $\boldsymbol{W}_2 \in \mathbb{R}^{c\times b}$:

(a) What is the Jacobian matrix of f and g

- Jacobian of f is W_1
- Jacobian of g is W_2

(b) What is the Jacobian matrix of $h(\boldsymbol{x}) = g(f(\boldsymbol{x})) = (g \circ f)(\boldsymbol{x})$

Jacobian of h is W_2W_1

(c) What is the Jacobian matrix of $h(\boldsymbol{x})$ if $\boldsymbol{W}_1 = \boldsymbol{W}_2$ (so $a = b = c$)

Jacobian of h is W_1W_1

1.3 Deriving Loss Functions

Derive the loss function for the following algorithms based on their common update rule $w_i \leftarrow w_i + \eta(y - \tilde{y})x_i$. Show the steps of the derivation given the following inference rules (simply stating the final loss function will receive no points).

1. (4 points) **Perceptron** $\tilde{y} = \text{sign}\left(b + \sum_{i=1}^{d} w_i x_i\right)$

   Update rule given: $w_i \leftarrow w_i + \eta(y - \tilde{y})x_i$.

   General rule: $w_i \leftarrow w_i + \eta\frac{\partial C}{\partial w_i}$.

   Comparing the two, we get: $\frac{\partial C}{\partial w_i} = -(y - \tilde{y})x_i$.

   $$C = \int \left(-(y - \tilde{y})x_i \frac{\partial w_i}{\partial w_i}\right) dw_i = \int \left(-(y - \text{sign}(b + \sum_{i=1}^{d} w_i x_i))x_i \frac{\partial w_i}{\partial w_i}\right) dw_i$$

   Now, if we treat $(y - \text{sign}(b + \sum_{i=1}^{d} w_i x_i))x_i$ as u, then the above term is a constant with respect to w_i. It is either $(y-1)x_i$ or $(y+1)x_i$, and thus, when we integrate with respect to w_i, we get the following:

   $$C = \begin{cases} -(y-1)\cdot x_i \cdot w_i & \text{if } b + \sum_{i=1}^{d} w_i x_i \geq 0 \\ -(y+1)\cdot x_i \cdot w_i & \text{if } b + \sum_{i=1}^{d} w_i x_i \leq 0 \end{cases}$$

   Thus,

   $$C_i = -(y - \tilde{y}) \cdot w_i \cdot x_i \implies C = -(y - \tilde{y}) \sum_{i=1}^{d} w_i \cdot x_i$$

2. (4 points) **Adaline / Least Mean Squares** $\tilde{y} = b + \sum_{i=1}^{d} w_i x_i$

   Update rule given: $w_i \leftarrow w_i + \eta(y - \tilde{y})x_i$.

   General rule: $w_i \leftarrow w_i + \eta \frac{\partial C}{\partial w_i}$.

   Comparing the two, we get: $\frac{\partial C}{\partial w_i} = -(y - \tilde{y})x_i$.

$$\frac{\partial C}{\partial w_i} = -(y - b - \sum_{i=1}^{d} w_i x_i)x_i$$

   Treating the term $-(y - b - \sum_{i=1}^{d} w_i x_i)$ as u,

$$\frac{du}{dw_i} = x_i \implies dw_i = \frac{du}{x_i}$$

   Substituting above, we get:

$$\frac{\partial C}{\partial w_i} = -(y - \tilde{y})x_i$$

$$C = \int -(y-\tilde{y})x_i \, dw_i = \int -u x_i \, du = \int -u du = -\frac{u^2}{2} = -\left(\frac{(y - b - \sum_{i=1}^{d} w_i x_i)^2}{2}\right) = -\frac{(y - \tilde{y})^2}{2}$$

3. (4 points) **Logistic Regression** $\tilde{y} = \tanh\left(b + \sum_{i=1}^{d} w_i x_i\right)$

   Update rule given: $w_i \leftarrow w_i + \eta(y - \tilde{y})x_i$.

   General rule: $w_i \leftarrow w_i + \eta \frac{\partial C}{\partial w_i}$.

   Comparing the two, we get: $\frac{\partial C}{\partial w_i} = -(y - \tilde{y})x_i$.

$$\frac{\partial C}{\partial w_i} = -(y - \tanh(b + \sum_{i=1}^{d} w_i x_i))x_i$$

   Treating the term $-(y - \tanh(b + \sum_{i=1}^{d} w_i x_i))$ as u,

$$\frac{du}{dw_i} = 1 - \tanh^2(b + \sum_{i=1}^{d} w_i x_i) \cdot x_i \implies dw_i = \frac{du}{1 - \tanh^2(b + \sum_{i=1}^{d} w_i x_i) \cdot x_i}$$

   Substituting above, we get:

$$\frac{\partial C}{\partial w_i} = -(y - \tilde{y})x_i$$

$$C = \int -(y-\tilde{y})x_i \, dw_i = \int u \, du \left(1 - \tanh^2(b + \sum_{i=1}^{d} w_i x_i) \cdot x_i\right)$$

$$C = \int \frac{u \, du}{1 - \tanh^2(b + \sum_{i=1}^{d} w_i x_i)}$$

Substituting $\tanh(b + \sum_{i=1}^{d} w_i x_i)$ as $y + u$ based on equation (i),

$$C = \int \frac{u \, du}{1 - (y+u)^2}$$

$$C = \frac{1}{2}((y-1)\log(1-u-y) - (y+1)\log(1+u+y))$$

Substituting u back, we get:

$$C = \frac{1}{2}((y-1)\log(1+\tilde{y}) - (y+1)\log(1-\tilde{y}))$$

2 Implementation (50pt)

2.1 Backpropagation (35pt)

You need to implement the forward pass and backward pass for `Linear`, `ReLU`, `Sigmoid`, `MSE loss`, and `BCE loss` in the attached `mlp.py` file. We provide three example test cases `test1.py`, `test2.py`, `test3.py`. We will test your implementation with other hidden test cases, so please create your own test cases to make sure your implementation is correct.

Recommendation: Go through this Pytorch tutorial to have a thorough understanding of Tensors.

Extra instructions:

1. We will put your `mlp.py` file under the same directory of the hidden test scripts and use the command `python hiddenTestScriptName.py` to check your implementation. So please make sure the file name is `mlp.py` and it can be executed with the example test scripts we provided.
2. You are not allowed to use PyTorch autograd functionality in your implementation.
3. Be careful about the dimensionality of the vector and matrix in PyTorch. It is not necessarily follow the the Math you got from part 1.

2.2 Gradient Descent (15pt + 5pt)

In DeepDream, the paper claims that you can follow the gradient to maximize an energy with respect to the input in order to visualize the input. We provide some code to do this. Given a image classifier, implement a function that performs optimization on the input (the image), to find the image that most highly represents the class. You will need to implement the `gradient_descent` function in `sgd.py`. You will be graded on how well the model optimizes the input with respect to the labels.

Extra hints:

1. We try to *minimize* the energy of the class, e.g. maximize the class logit. Make sure you are following the gradient in the right direction
2. A reasonable starting learning rate to try is 0.01, but depending on your implementation, make sure to sweep across a few magnitudes.
3. Make sure you use `normalize_and_jitter`, since the neural network expect a normalized input. Jittering produces more visually pleasing results

You may notice that the images that you generate are very messy and full of high frequency noise. Extra credit (5 points) can be had by generating visually pleasing images (it is really really hard to make it work!), and experimenting with visualizing the middle layers of the network. There are some tricks to this:

1. Blur the image at each iteration, which reduces high frequency noise
2. Clamp the pixel values between 0 and 1
3. Implement weight decay
4. Blur the gradients at each iteration
5. Implement gradient descent at multiple scales, scaling up every so often
6. add gradient clipping to prevent the exploding gradient.