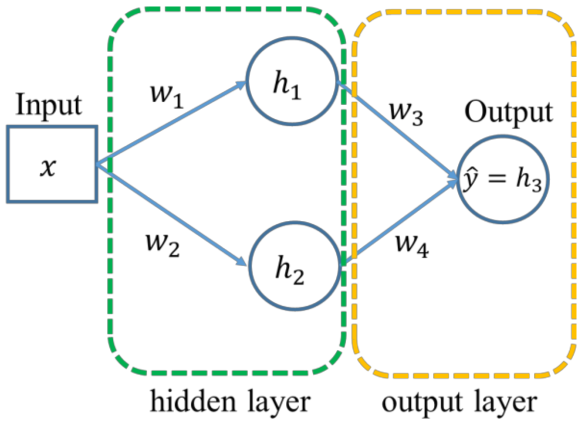
Homework Assignment 4

Part-1: Basic Concepts

1. Backpropagation in A Neural Network (14 points)



$$h_1 = f_1(w_1 x + b_1)$$

$$h_2 = f_2(w_2 x + b_2)$$

$$\hat{y} = f_3(w_3 h_1 + w_4 h_2 + b_3)$$

$$f_n' = \frac{\partial f_n(v)}{\partial v}, \text{ n} = 1,2,3$$

$x, w_1, w_2, w_3, w_4, b_1, b_2, b_3, h_1, h_2, h_3$ are scalars

Compute the derivatives of the loss L with respect to parameters and input, assuming $\frac{\partial L}{\partial h_3}$ is known.

Example:

This is the complete solution: $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial w_1} = \frac{\partial L}{\partial h_3} f_3' w_3 f_1' x$ and you get 2 points.

This is a partial solution: $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial w_1}$, and you get 1 point.

Each of these derivatives is worth 2 points

$\frac{\partial L}{\partial w_2}$

$\frac{\partial L}{\partial w_3}$

$\frac{\partial L}{\partial w_4}$

$\frac{\partial L}{\partial b_1}$

$\frac{\partial L}{\partial b_2}$

$\frac{\partial L}{\partial b_3}$

$\frac{\partial L}{\partial x}$

2. Computational Graph (6 points)

$h = 2x + 1$

$z = x^2 + h^2$

$y = \dfrac{1}{1 + e^{-h}}$

(1: 1 point) Draw the computational graph based on the above three lines of code

(2: 5 points) What is the value of $\frac{\partial y}{\partial z}$ according to the computational graph ? (not pure math)

3. Why are Nonlinear Activation Functions Needed in Neural Networks ? (5 points)

(1: 1 point) Assume that we have developed an MLP. Inside the MLP, there are *N* linear layers (including hidden layers and output layer), and each of these layers is followed by ReLU activation. Now, we remove those ReLU activations from the MLP. Show that the MLP without ReLU activations is equivalent to a simple linear transform of the input, i.e., $y = Wx$

Hint:

Linear Layer-1: $y_1 = W_1 x$ (note: it can also be written as $y_1 = W_1^T x$)
Linear Layer-1 Followed by ReLU activation: $y_1 = ReLU(W_1 x)$

(2: 4 point) Assume that we have developed a 1D CNN. This CNN has *N* convolution layers (including hidden layers and output layer), and each of these layers is followed by ReLU activation. Now, we remove those ReLU activations from the CNN. Show that the CNN without ReLU activations is equivalent to a simple linear transform of the input, i.e., $y = Wx$

Hint: Can a convolution layer be written as a linear transform? If so, how? Read the lecture notes

4. Target (output) Normalization for a Neural Network (2 points)

Usually, we need to apply normalization/standardization to the inputs for classification and regression tasks, so that the input will be in the range of 0 to 1, or -1 to +1. For example, if the input is an image, then every pixel value is divided by 255, so that the pixel values of the normalized image are in the range of 0 to 1. Input normalization facilitates the convergence of training algorithms.

Do we need to apply normalization to the output ? To answer this question, let's consider the following example: assume the input is an image of a person, the output vector from a neural network has two components, $\hat{y}_{(1)}$ and $\hat{y}_{(2)}$: $\hat{y}_{(1)}$ is the monthly income (in the range of 0 to 10,000), and $\hat{y}_{(2)}$ is the age (in the range of 0 to 100). The MSE loss for a single data sample is defined as

$$L = (\hat{y}_{(1)} - y_{(1)})^2 + (\hat{y}_{(2)} - y_{(2)})^2$$

where $y_{(1)}$ and $y_{(2)}$ are ground truth values of an input data sample.

(2 points): is output (i.e., the output target $y_{(1)}, y_{(2)}$) normalization necessary for this task? Why? If it is necessary, what kind of normalization can be applied ?

5. Activation Functions for Regression (3 points)

Neural networks can be used for regression. To model nonlinear input-output relationship, a neural network needs nonlinear activation functions in the hidden layers. Usually, the output layer does not need nonlinear activation functions. However, sometimes, there are requirements for outputs. For example, if the output is the sale price of a house, then the output should be nonnegative.

Assume $\boldsymbol{z}$ is the scalar output of a network, and the network does not have nonlinear activation function in the output layer. Now, there is some requirement for output, and you decide to add a nonlinear activation function.

You design nonlinear activation functions for three different requirements:

(1: 1 point) the final output y should be nonnegative ($y \geq 0$), then what is the activation function $y = f(z)$?

(2: 1 point) the final output y should be nonpositive ($y \leq 0$), then what is the activation function $y = f(z)$?

(3: 1 point) the final output y should be $a \leq y \leq b$, then what is the activation function $y = f(z)$?

The above activation functions should be based on activation functions in PyTorch: choose appropriate activation functions or combine some activation functions or modify some activation functions available in PyTorch.
https://pytorch.org/docs/stable/nn.functional.html#non-linear-activation-functions
Do NOT use if statements, for example:

```
def activation(z):
    if z < a:     return a
    elif z > b:   return b
    else: return z
```

The above activation function is not an acceptable answer.

6. ReLU and Piecewise Linear (5 points)

(1: 4 points) Prove that an MLP with ReLU activations is a piecewise linear function of the input

(1: 1 points) Prove that a 1D CNN with ReLU activations is a piecewise linear function of the input

Note: this is not the answer "it is because ReLU is piecewise linear"

Note: it is also true that 2D/3D CNN with ReLU activations is a piecewise linear function of the input

Part-2 Programming

Programming tasks: H4P2T1.ipynb and H4P2T2.ipynb. Read H4P2_torchview.ipynb

Grading : The number of points for each question/task

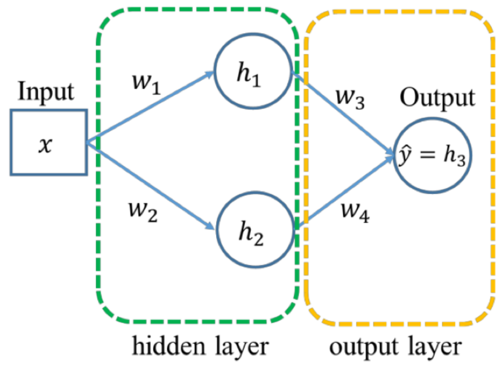
	Undergrad Student	Graduate Student
1. Backpropagation	14	14
2. Computational Graph	6	6
3. Nonlinear Activations in NN	extra 5 points (bonus)	5
4. Output Target Normalization	2	2
5. Activations for Regression	3	3
6. ReLU and piecewise linear	N.A.	extra 5 points (bonus)
H4P2T1 (MLP)	25	20
H4P2T2 (CNN)	50	50
Total	100 + 5	100 + 5

Read the instructions about H4P2T1 and H4P2T2 on the following pages.

Homework 4

Sloan Atkins
CSC 546
Fall 25

1. Backpropagation in A Neural Network (14 points)



$h_1 = f_1(w_1 x + b_1)$

$h_2 = f_2(w_2 x + b_2)$

$\hat{y} = f_3(w_3 h_1 + w_4 h_2 + b_3)$

$f_n' = \frac{\partial f_n(v)}{\partial v}$, n = 1,2,3

$x, w_1, w_2, w_3, w_4, b_1, b_2, b_3, h_1, h_2, h_3$ are scalars

Compute the derivatives of the loss L with respect to parameters and input, assuming $\frac{\partial L}{\partial h_3}$ is known.

Example:

This is the complete solution: $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial w_1} = \frac{\partial L}{\partial h_3} f_3' w_3 f_1' x$ and you get 2 points.

This is a partial solution: $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial w_1}$, and you get 1 point.

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w_2} = \frac{\partial L}{\partial h_3} f_3' w_4 f_2' x$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial w_3} = \frac{\partial L}{\partial h_3} f_3' h_1$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial w_4} = \frac{\partial L}{\partial h_3} f_3' h_2$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial b_1} = \frac{\partial L}{\partial h_3} f_3' w_3 f_1'$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial b_2} = \frac{\partial L}{\partial h_3} f_3' w_4 f_2'$$

$$\frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial b_3} = \frac{\partial L}{\partial h_3} f_3'$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial x} + \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial x}$$
$$= \frac{\partial L}{\partial h_3} f_3' w_3 f_1' w_1 + \frac{\partial L}{\partial h_3} f_3' w_4 f_2' w_2$$

2. Computational Graph (6 points)

$h = 2x + 1$

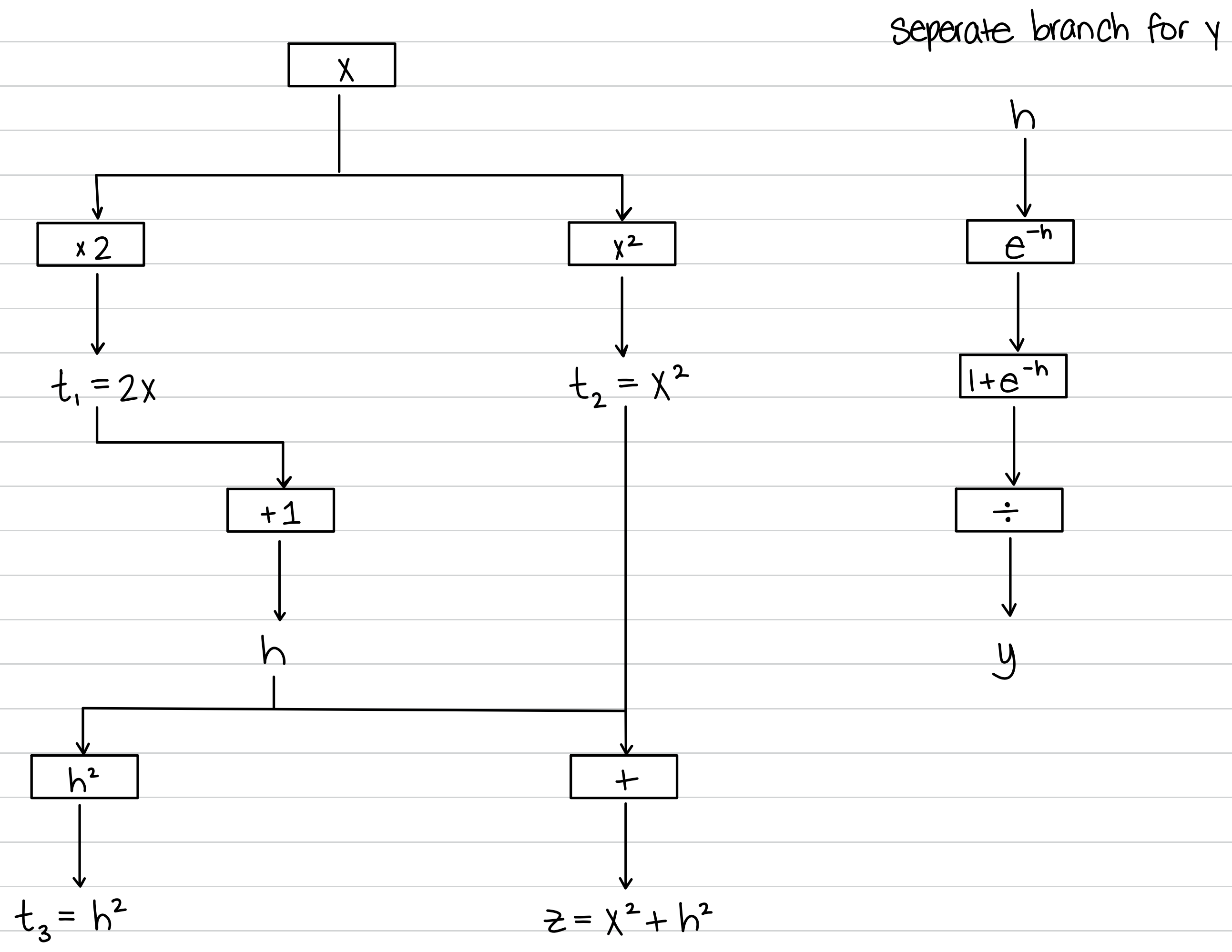
$z = x^2 + h^2$

$y = \frac{1}{1+e^{-h}}$

(1: 1 point) Draw the computational graph based on the above three lines of code

(2: 5 points) What is the value of $\frac{\partial y}{\partial z}$ according to the computational graph ? (not pure math)

1)



2) According to the computational graph, there is no path from z to y, so changing z cannot effect y, which means $\frac{\partial y}{\partial z} = 0$

Rule in computational graphs:
- If no arrow goes from node to the output, its gradient is zero.

3. Why are Nonlinear Activation Functions Needed in Neural Networks ? (5 points)

(1: 1 point) Assume that we have developed an MLP. Inside the MLP, there are N linear layers (including hidden layers and output layer), and each of these layers is followed by ReLU activation. Now, we remove those ReLU activations from the MLP. Show that the MLP without ReLU activations is equivalent to a simple linear transform of the input, i.e., $y = Wx$

Hint:

Linear Layer-1: $y_1 = W_1 x$ (note: it can also be written as $y_1 = W_1^T x$)
Linear Layer-1 Followed by ReLU activation: $y_1 = ReLU(W_1 x)$

(2: 4 point) Assume that we have developed a 1D CNN. This CNN has N convolution layers (including hidden layers and output layer), and each of these layers is followed by ReLU activation. Now, we remove those ReLU activations from the CNN. Show that the CNN without ReLU activations is equivalent to a simple linear transform of the input, i.e., $y = Wx$

Hint: Can a convolution layer be written as a linear transform? If so, how? Read the lecture notes

1) A linear layer has the form: $y_1 = W_1 x$

If we stack N linear layers without ReLU, we get:
$$y_N = W_N(W_{N-1}(\dots W_2(W_1 x)))$$

Matrix multiplication is associative, so:
$$y_N = (W_N W_{N-1} \dots W_2 W_1)x$$

Let: $W = W_N W_{N-1} \dots W_1$

Then: $y = Wx$

∴ An MLP without nonlinear activations is just a single linear transformation

2) A convolution layer is also a linear operation
It can be written as: $y = Kx$
where K is a Toeplitz matrix formed from the convolution Kernel

Stacking N convolution layers without ReLU:
$$y_N = K_N(K_{N-1}(\dots K_2(K_1 x)))$$

Because convolution is linear:
$$y_N = (K_N K_{N-1} \dots K_1)x$$

Let: $W = K_N K_{N-1} \dots K_1$
Then: $y = Wx$

∴ A CNN without nonlinear activations is just a linear transform of the input
Even though convolution looks different from a fully connected layer, both are linear.
Removing ReLU eliminates all nonlinearities, so the entire CNN reduces to:
$$y = Wx$$

4. Target (output) Normalization for a Neural Network (2 points)

Usually, we need to apply normalization/standardization to the inputs for classification and regression tasks, so that the input will be in the range of 0 to 1, or -1 to +1. For example, if the input is an image, then every pixel value is divided by 255, so that the pixel values of the normalized image are in the range of 0 to 1. Input normalization facilitates the convergence of training algorithms.

Do we need to apply normalization to the output ? To answer this question, let's consider the following example: assume the input is an image of a person, the output vector from a neural network has two components, $\hat{y}_{(1)}$ and $\hat{y}_{(2)}$: $\hat{y}_{(1)}$ is the monthly income (in the range of 0 to 10,000), and $\hat{y}_{(2)}$ is the age (in the range of 0 to 100). The MSE loss for a single data sample is defined as

$$L = (\hat{y}_{(1)} - y_{(1)})^2 + (\hat{y}_{(2)} - y_{(2)})^2$$

where $y_{(1)}$ and $y_{(2)}$ are ground truth values of an input data sample.

(2 points): is output (i.e., the output target $y_{(1)}$, $y_{(2)}$) normalization necessary for this task? Why? If it is necessary, what kind of normalization can be applied ?

Yes, output normalization is necessary

Reasons why:

- The two targets have very different scales:
  - Income: 0 to 10,000
  - Age: 0 to 100
- In MSE loss, big numbers dominate the loss
- So the network will focus almost entirely on predicting income and largely ignore age because:

$$(\hat{y}_{(1)} - y_{(1)})^2 \gg (\hat{y}_{(2)} - y_{(2)})^2$$

- This slows training and produces a biased model

To fix this, each output can be normalized independently

<u>Option 1</u>: Min-max normalization

Normalize each target to [0,1]: $y' = \dfrac{y - y_{min}}{y_{max} - y_{min}}$

Example:

- Income: divide by 10,000
- Age: divide by 100

<u>Option 2</u>: Standardization (z-score)

$$y' = \frac{y - \mu}{\sigma}$$

Where μ and σ are computed from dataset

5. Activation Functions for Regression (3 points)
Neural networks can be used for regression. To model nonlinear input-output relationship, a neural network needs nonlinear activation functions in the hidden layers. Usually, the output layer does not need nonlinear activation functions. However, sometimes, there are requirements for outputs. For example, if the output is the sale price of a house, then the output should be nonnegative.
Assume $\boldsymbol{z}$ is the scalar output of a network, and the network does not have nonlinear activation function in the output layer. Now, there is some requirement for output, and you decide to add a nonlinear activation function.

You design nonlinear activation functions for three different requirements:

(1: 1 point) the final output y should be nonnegative ($y \geq 0$), then what is the activation function $y = f(z)$?

(2: 1 point) the final output y should be nonpositive ($y \leq 0$), then what is the activation function $y = f(z)$?

(3: 1 point) the final output y should be $a \leq y \leq b$, then what is the activation function $y = f(z)$?

The above activation functions should be based on activation functions in PyTorch: choose appropriate activation functions or combine some activation functions or modify some activation functions available in PyTorch.
https://pytorch.org/docs/stable/nn.functional.html#non-linear-activation-functions
Do NOT use if statements, for example:

```
def activation(z):
    if z < a:     return a
    elif z > b:   return b
    else: return z
```

The above activation function is not an acceptable answer.

1) Best activation · ReLU
$y = \text{ReLU}(z)$

Because ReLU outputs:
$\text{ReLU}(z) = \max(0, z) \geq 0$

2) Using negative ReLU:
$y = -\text{ReLU}(z)$

$\text{ReLU}(z) \geq 0$, so multiplying by -1 gives: $y \leq 0$

3) Use Sigmoid scaled to the interval
Sigmoid outputs values in $(0,1)$
We transform it to $[a,b]$:
$y = a + (b-a)\sigma(z)$

Where: $\sigma(z) = \dfrac{1}{1+e^{-z}}$

This matches the requirement: $a \leq y \leq b$

6. ReLU and Piecewise Linear (5 points)

(1: 4 points) Prove that an MLP with ReLU activations is a piecewise linear function of the input

(1: 1 points) Prove that a 1D CNN with ReLU activations is a piecewise linear function of the input

Note: this is not the answer "it is because ReLU is piecewise linear"

Note: it is also true that 2D/3D CNN with ReLU activations is a piecewise linear function of the input

(1) Prove an MLP with ReLU activations is piecewise linear

Each ReLU introduces a "case split":

$$\mathrm{ReLU}(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$$

For each ReLU nit, the output is either:
- active → behaves like a linear function z
- inactive → output is 0 (also linear)

Also many ReLUs, each pattern of "on/off" activations defines one linear region

Proof:
An MLP layer is: $h^{(k)} = \mathrm{ReLU}(W_k h^{(k-1)} + b_k)$

Consider a fixed input x
For each neuron in each layer, the pre-activation:

$$z_i = W_i x + b_i$$

is either:
- positive, so ReLU outputs z_i (a linear function), or
- non-positive, so ReLU outputs 0 (also linear)

Thus for that region of input space, the ReLU pattern is fixed
This means the network becomes:

$$y = Ax + c,$$

where A and c depend on which ReLUs are "on" and "off"

Since there are finitely many ReLUs, there are finitely many such patterns
Therefore, the model is: a finite set of linear functions, each valid on one region of the input space.
↪ Piecewise linear function definition

(2) Prove a 1D CNN with ReLU is piecewise linear

A convolution is a linear transform: $h = Kx$
where K is a Toeplitz matrix built from the Kernel

So a CNN layer is: $y = \text{ReLU}(Kx + b)$

For any fixed input region, each ReLU is either:
- active → behaves like
- inactive ⇒ output is 0 (linear)

Thus, for a fixed pattern of ReLU activations:
$$y = Ax + c$$
which is linear

Across the whole input space, these activation patterns change, producing a finite number of linear regions

Thus, a 1D CNN with ReLU is also a piecewise linear function of the input