Week 3 - Neural Networks I (Forward Calculation)

□□ Overview

This week we will implement the forward pass of a Neural Network classifier from scratch. It is always a good practice to do it yourself for a new algorithm that you don't deeply understand, so that you learn how it works.

Two tasks for this week. You will:

1. implement the forward pass through the network, use weights that have already been learnt, and perform basic prediction/inference and evaluation of the network.
2. read and understand the re-implementation by using pytorch.

Problem Context

This network will be used to classify people as having either **benign or malignant breast tumours**.

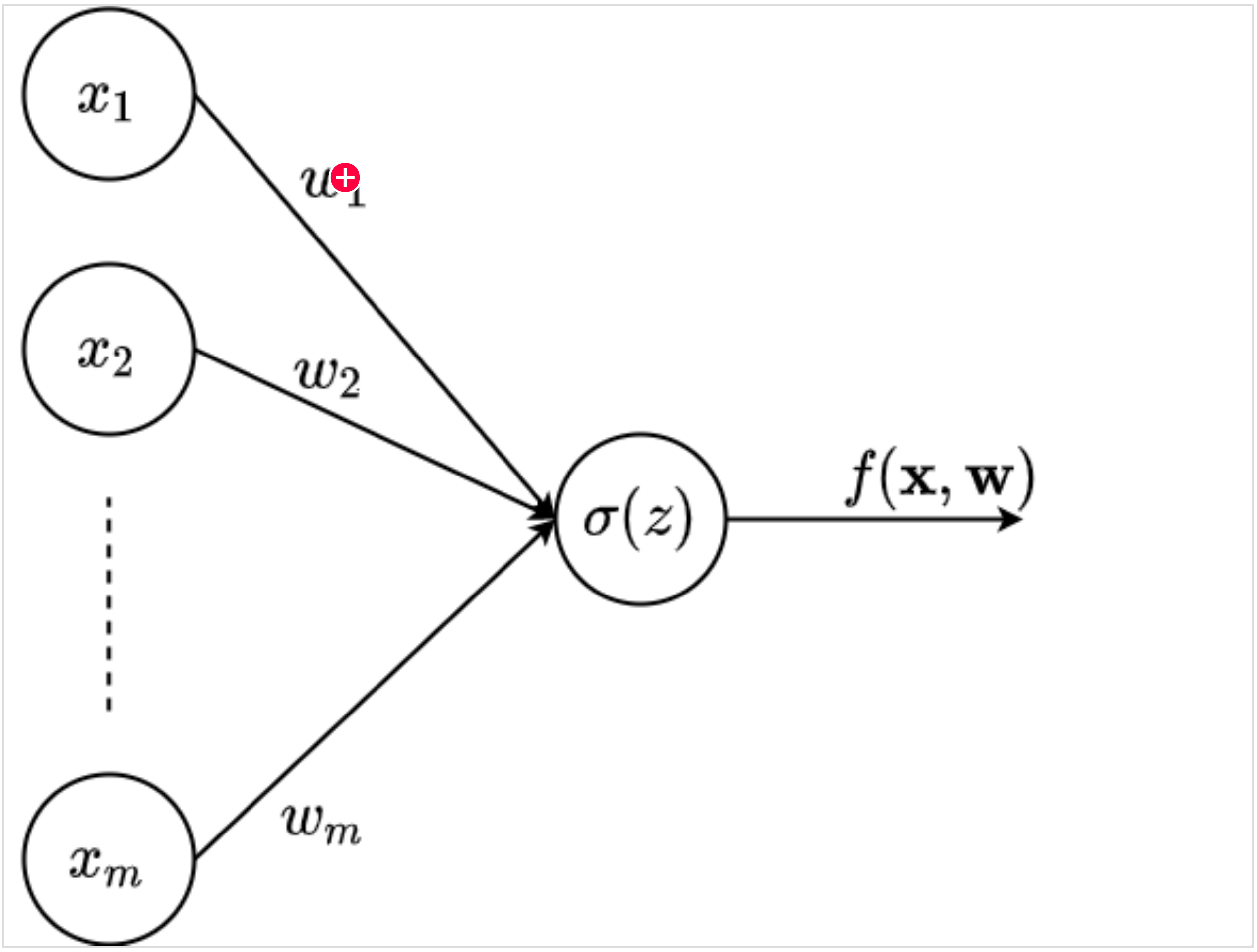
The breast cancer dataset has the following attributes:

- Two classes (binary)
- 357 benign cases (0) and 212 malignant (cancer) cases (1)
- 30 features

󠀠󠀠 Warmup Questions

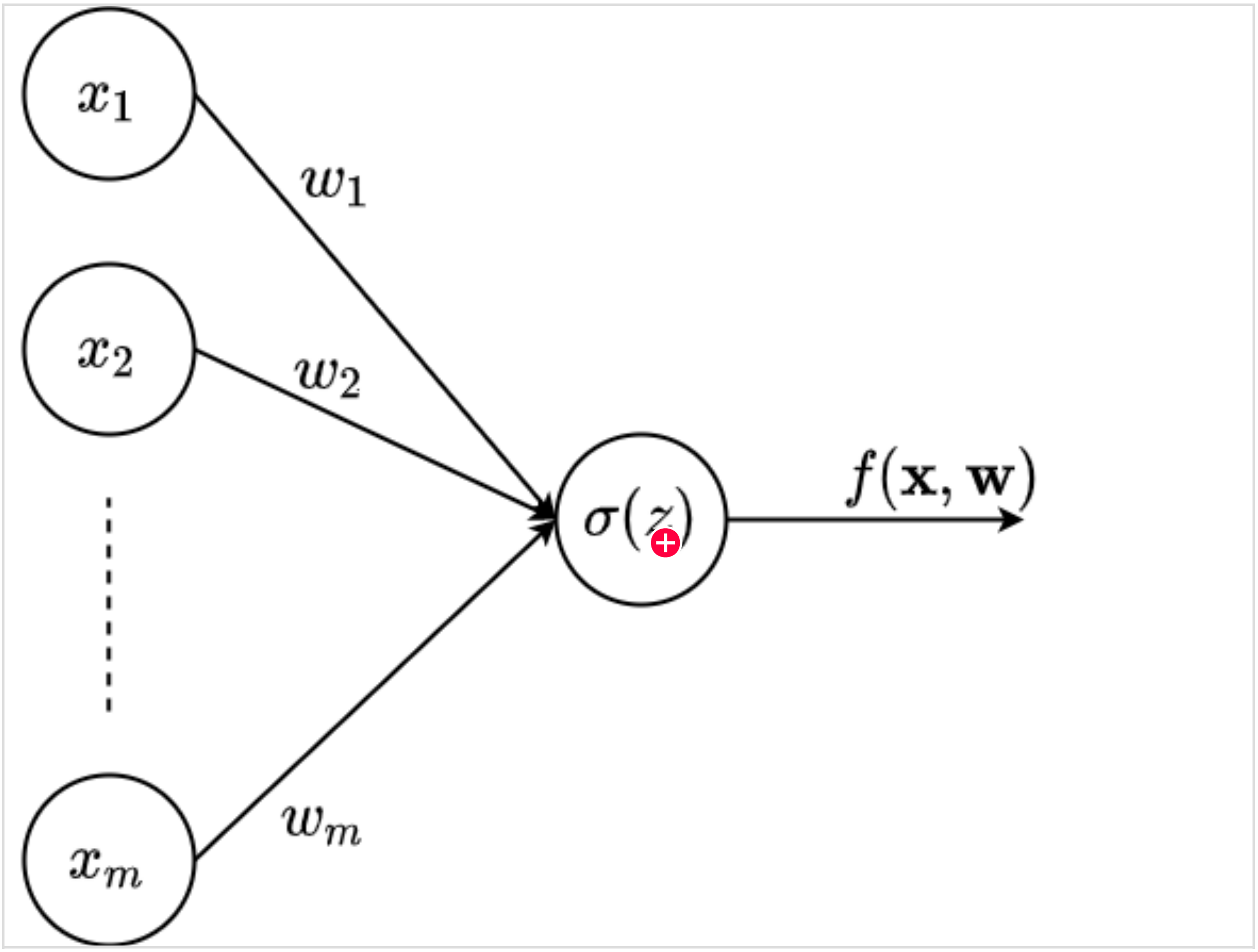
Question 1 *Submitted Mar 9th 2022 at 11:17:23 am*

On the diagram mark where the weights are represented.



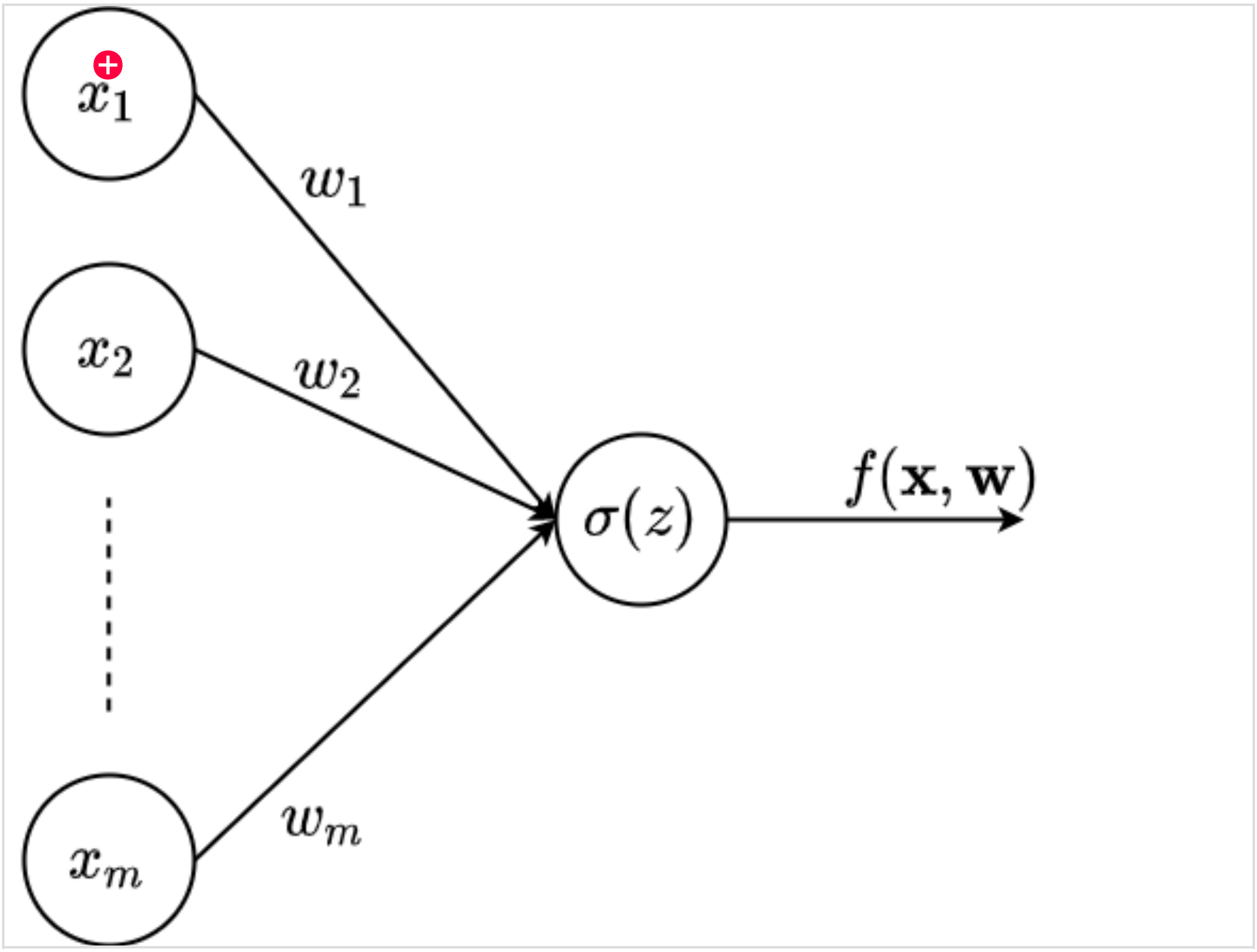
Question 2 *Submitted Mar 9th 2022 at 11:17:13 am*

On the diagram mark where the activation function is represented.



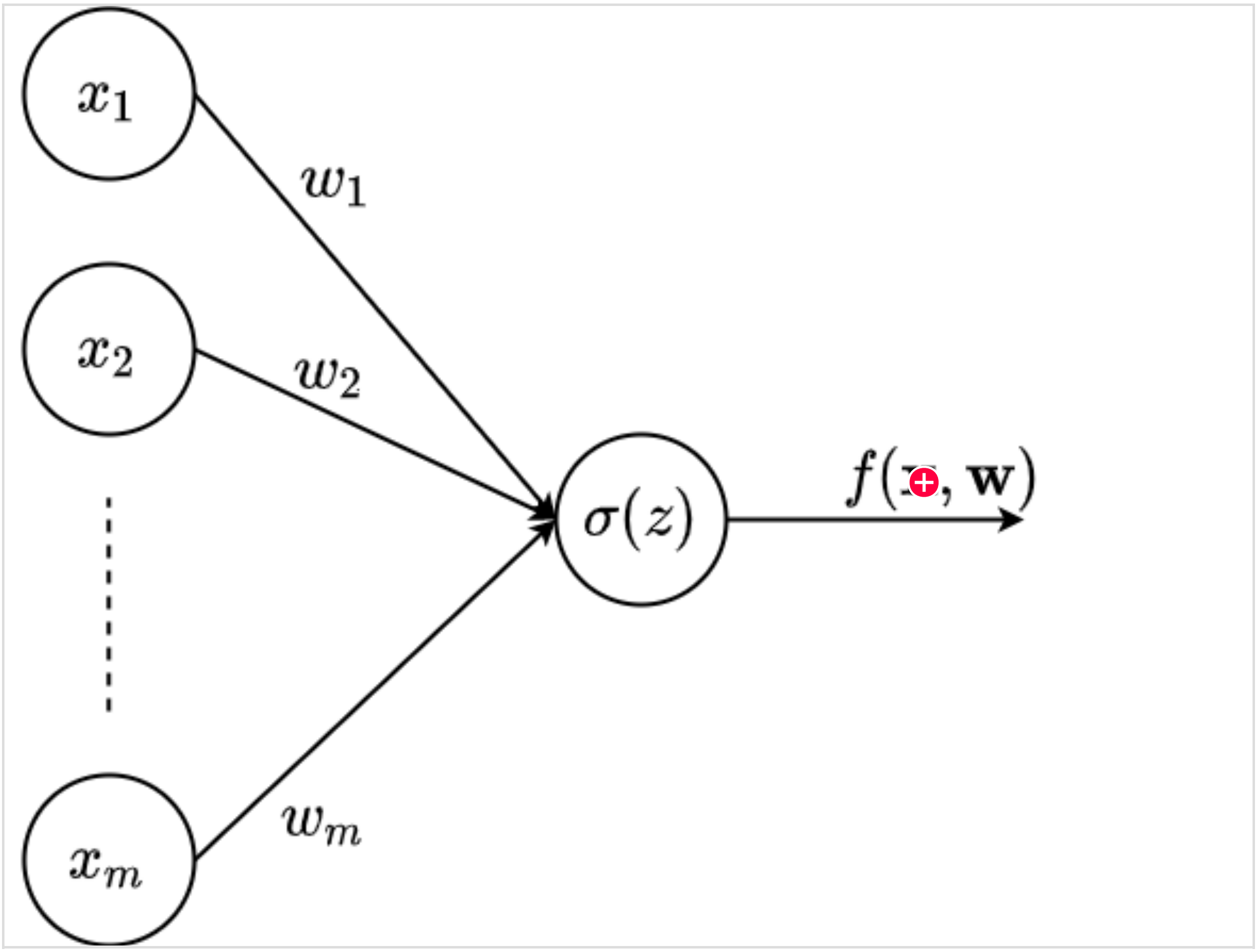
Question 3 *Submitted Mar 9th 2022 at 11:17:06 am*

On the diagram mark where the input features are represented.



Question 4 *Submitted Mar 9th 2022 at 11:17:00 am*

On the diagram mark where the output is represented.



Question 5 *Submitted Mar 9th 2022 at 11:16:52 am*

Given the network below, order the code so that it prints the forward pass (prediction). The activation will be Sigmoid function.

Drag from here

```
w = np.array([[4.6, 0.4, -0.8, 0.3]])
```

```
f_xw = np.maximum(a, 0)
```

```
import numpy as np
x = np.array([[0.1, 2.3, -0.5, 10]])
```

```
f_xw = a
```

```
a = x * w
```

Drop blocks here

```
import numpy as np
x = np.array([0.1, 2.3, -0.5, 10])
```

```
w = np.array([4.6, 0.4, -0.8, 0.3])
```

```
a = np.dot(x, w)
```

```
f_xw = 1/(1+np.exp(-a))
```

```
print(f_xw)
```

□□ Forward Pass Exercises

Question 1 *Submitted Mar 9th 2022 at 11:23:54 am*

Using the activation function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

and weights and data

$$\mathbf{w} = [0.2, -0.7, 0.5]^T$$

$$\mathbf{x} = [1.2, 3.4, -0.5]^T$$

Calculate the output $f(\mathbf{x}, \mathbf{w})$ to 2 decimal places.

0.08

Question 2 *Submitted Mar 9th 2022 at 11:29:24 am*

Use the ReLU activation function

$$\sigma(z) = \max(0, z)$$

Using the same weights and data, calculate the output to 2 decimal places.

0

Question 3 *Submitted Mar 9th 2022 at 11:36:50 am*

Consider a three-layer NN as follows. Use the Sigmoid activation function for both hidden layer and output layer, with the following weights and data. Calculate the output to 2 decimal places.

$$\mathbf{x} = [2.1, 4.3, 0.5]^T$$

$$\mathbf{w}^{(1)} = [[-0.8, -0.1, 3], [0.5, 4.1, -0.5]]^T$$

$$\mathbf{w}^{(2)} = [3.1, -2]^T$$

0.29

□□ Network Architecture

The network we will use consists of the following:

- Input layer with fully connected bias unit
- Hidden layer with 10 neurons using **linear** activation and a bias unit
- Output layer using **sigmoid** activation

Maths

Layer 1 (Input Layer)

$$\mathbf{a}^{(1)} = \mathbf{x}$$

Layer 2 (Hidden Layer)

[Note: In lecture note, the bias $\mathbf{b}^{(1)}$ was not considered]

$$\mathbf{a}^{(2)} = \sigma_1(\mathbf{a}^{(1)}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

where

- $\mathbf{x}$ is the 1×30 vector of features
- $\mathbf{W}^{(1)}$ is the 30×10 matrix of layer 1 weights
- $\mathbf{b}^{(1)}$ is the 10 element vector of bias weights
- $\sigma_1(z) = z$ is the linear activation function
- $\mathbf{a}^{(1)}$ is 1×10 row vector result of layer 1

Layer 3 (Output Layer)

$$\hat{y} = a^{(3)} = \sigma_2(\mathbf{a}^{(2)}\mathbf{w}^{(2)} + b^{(2)})$$

where

- $\hat{y}$ is the scalar network output (prediction)
- $\sigma_2(z) = \frac{1}{1+e^{-z}}$ is the sigmoid activation function
- $\mathbf{w}^{(2)}$ is the 10×1 vector of weights
- $b^{(2)}$ is the bias weight

Diagram

□□ Task 1: Define the Model

Write a Python class that **implements the forward pass** the Neural Network defined on the previous slides.

Example Usage

```
model = NeuralNetwork(True)
model.predict_proba(X)
model.predict(X)
```

Formulas [We skip the input layer as there is no operation on it.]

$$\mathbf{a}^{(2)} = \sigma_1(\mathbf{x}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

$$\hat{y} = a^{(3)} = \sigma_2(\mathbf{a}^{(2)}\mathbf{w}^{(2)} + b^{(2)})$$

Class Specification

- name: `NeuralNetwork`
- methods:
    - `__init__`
    - `predict_proba`
        - parameters:
            - `X` (numpy array): observations with shape (n_samples, n_features)
        - returns:
            - `y` (numpy array): probability for each sample in X
    - `predict`
        - parameters:
            - `X` (numpy array): observations with shape (n_samples, n_features)
        - returns:
            - `y` (numpy array): predicted class for each sample in X

Instructions and Hints

1. The weights for the network are stored in the `parameters.json` file
2. Name the weights and bias arrays: "W1", "b1", "W2" and "b2"
3. The `predict` method should call the `predict_proba` method to avoid repeated code

□□ Try the Model

Using the model you defined earlier, print the

- probability of malignant cancer
- cancer classification (either 0 for benign or 1 for malignant)

for the first patient in the test set.

Expected Output

```
0.04XXXXXXXXXXXXXX
0
```

Questions:

1. Does the predicted class match the predicted classification?
2. What probability value is required for a malignant prediction?

▯▯ Confusion Matrix

Using the model you defined earlier, print the confusion matrix over the entire test set.

Expected Output

```
[[XX  X]
 [ X XX]]
```

Hints and Tips

1. Click here for the docs of `confusion_matrix`

□□ Classification Report

Using the model you defined earlier, print the classification report over the entire test set.

Expected Output

```
              precision    recall  f1-score   support

         0.0       0.XX      0.XX      0.XX        78
         1.0       0.XX      0.XX      0.XX        65

    accuracy                           0.XX       143
   macro avg       0.XX      0.XX      0.XX       143
weighted avg       0.XX      0.XX      0.XX       143
```

Hints and Tips

1. Click here for the docs of `classification_report`

□□ Follow Up Questions

Question 1 *Submitted Mar 9th 2022 at 1:01:36 pm*

Check the size of each class. Is this a balanced or imbalanced classification task? Relating to the context of the problem, should you focus on reducing the rate of FP or FN?

imbalanced

Question 2 *Submitted Mar 9th 2022 at 1:01:30 pm*

Can you think of any reason(s) why our neural network wasn't able to achieve perfect predictions?

The very most disadvantage of a neural network is its black box nature. Because it has the ability to approximate any function, study its structure but don't give any insights on the structure of the function being approximated.

Question 3 *Submitted Mar 9th 2022 at 12:59:51 pm*

What do you think could be done to potentially improve the predictions?

Increase hidden Layers.
Change Activation function.
Change Activation function in Output layer.
Increase number of neurons.
Weight initialization.
More data.
Normalizing/Scaling data.

Task 2: Model in Pytorch

In Task 1, you implemented a neural network classifier forward pass from scratch. You understand what we need for the network. In this task, we will re-implement the same network using Pytorch.

First we introduce you to some neural network basics in Pytorch.

Linear layers

Linear Layer is the building block of a neural network model: $\mathbf{y} = \mathbf{X}\mathbf{W} + \mathbf{b}$. Documentation: https://pytorch.org/docs/1.9.1/generated/torch.nn.Linear.html.

When you create the linear layer (a module in torch term), the weights and bias terms are automatically initialized as parameters (torch tensor that requires gradient). The values are randomly initialized. You can check the current values by `layer.weight` and `layer.bias.`

```
import torch
torch.manual_seed(0)

# linear layer from 10 to 5 features with bias (default=True)
mylinear = torch.nn.Linear(10, 5, bias = True)

# applied on some data
N = 3
d = 10
x = torch.randn(N,d)
print(mylinear(x))
print()

print(mylinear.weight)
print(mylinear.bias)
```

We can set the values of the weights to the layer similarly as before

```
import torch

mylinear = torch.nn.Linear(10, 5, bias = True)

# predefined weight and bias values
weight = torch.zeros(10, 5)
bias = torch.zeros(5)

# set the values!!
mylinear.weight.data = weight
mylinear.bias.data = bias
```

```
# check the value of weights and bias now
print(mylinear.weight)
print(mylinear.bias)
```

Activation

Activation can also be used as a layer, which is applied elementwise on the input.

```
import torch
torch.manual_seed(0)

myrelu = torch.nn.ReLU()
mysigmoid = torch.nn.Sigmoid()
mytanh = torch.nn.Tanh()

# try applying activation on the data x below
N = 3
d = 5
x = torch.randn(N,d)
print(x)

# apply relu to x
print(myrelu(x))

# apply sigmoid to x
print(mysigmoid(x))

# apply tanh to x
print(mytanh(x))
```

Loss Functions

There are many loss functions available in Pytorch. Most common ones are MSE for regression and cross entropy for classification.

```
import torch
torch.manual_seed(0)

myMSE = torch.nn.MSELoss()
myBCE = torch.nn.BCELoss() # binary cross entropy for binary classification
myCE = torch.nn.CrossEntropyLoss() # cross entropy for multiclass classification

# calculate BCE for the following prediction
y_true = torch.empty(3).random_(2)
y_pred = torch.sigmoid(torch.randn(3))
print(y_true)
print(y_pred)
print(myBCE(y_pred, y_true))
```

Optimizers

There are many optimizers to choose from. We have seen Gradient Descent (`GD`) and Stochastic Gradient Descent (`SGD`) in Week 2. In practice, we often use `SGD` rather than `GD` because we have many samples. There are some faster variants of `SGD`, including `Adam`, `RMSProp`, `AMSGrad`, etc.

```
import torch

# we need to pass in a list of parameters to optimizers
W = torch.nn.Parameter(torch.zeros(3,2))
optimizer = torch.optim.SGD([W], lr=1e-5)
```

For this week, we do not train the weights of neural network (backward pass). We focus on how the output is calculated given the values of weights (forward pass). Thus, the loss function and optimizer are not used.

Neural Network Model Class

Pytorch allows you to define your own neural network model class. The following example defines a NN with two layers (input and output layers). The output layer uses Sigmoid activation function.

Comments:

1. All neural networks you define should be a *subclass* of `torch.nn.Module.` This allows all the properties of Pytorch model, such as automatic differentiation to be carried over to your self-defined model.
2. In the `__init__` function, we need to add a line `super().__init__().` This is to inherit the properties and attributes of the parent class `torch.nn.Module` to your model.
3. The `forward` function is the main function where you compute the output given the input.
4. We don't need to define the backward function, because Pytorch can do automatic gradient calculation.

```
import torch

class myNeuralNetwork(torch.nn.Module):
    def __init__(self, indim):
        super().__init__()
        self.Linear = torch.nn.Linear(indim, 1)
        self.Sigmoid = torch.nn.Sigmoid()
    def forward(self, x):
        x = self.Linear(x)
        x = self.Sigmoid(x)
        return x
```

```

```

Feedback Survey

Please give the teaching team some feedback for this week!

Question 1 *Submitted Mar 9th 2022 at 12:52:42 pm*

How did you feel after the tutorial?

- Great!
- [✓ Satisfied]
- Disappointed
- Confused

Question 2 *Submitted Mar 9th 2022 at 12:52:47 pm*

How do you feel about the speed of the tutorial?

- Too fast
- [✓ Just right]
- Too slow

Question 3 *Submitted Mar 9th 2022 at 12:52:49 pm*

How do you feel about the difficulty of concepts?

- [✓ Too hard]
- Just right

○ Too easy

Question 4 *Submitted Mar 9th 2022 at 12:52:51 pm*

The material was clear and easy to understand

○ True

[✓ False]

Question 5 *Submitted Mar 9th 2022 at 12:52:53 pm*

The tutor was clear and easy to understand

[✓ True]

○ False

Question 6 *Submitted Mar 9th 2022 at 12:52:54 pm*

Please write any general comments, feedback or elaborate on your previous answers.

Good