CS167: Machine Learning

Modular Implementation of Multilayer Perceptron (MLP) with PyTorch

Monday, April 13th, 2026

Drake UNIVERSITY

Recap: Important Design Questions for MLP

- Each of these questions need to be answered before you set up your **multilayer perceptron**
  - Q1: how many hidden layers should be there? (depth)
  - Q2: how many neurons should be in each layer? (width)
  - Q3: how many dense connections should be there in between each adjacent layers
  - Q4: what should the activation be at each of the intermediate layers?
    - `sigmoid(), tanh(), rectified-linear-unit(), etc`
  - Q5: what should be activation of the final layer
    - `depends the task` *`classification`* `(sigmoid(), softmax()) vs.` *`regression`*

List of PyTorch Functions We Need

- nn.Linear()
  creates the dense connections between two adjacent layers (*left layer* and *right layer*)
  just provide **#neurons_left_layer** and **#neurons_right_layer**
- nn.ReLU()
- nn.Softmax()
- nn.flatten()
- nn.Sequential()

- Let's jump into the notebook for a detailed discussion
  - https://github.com/alimoorreza/CS167-sp26-notes/blob/main/Day20_MLP_with_PyTorch.ipynb

Open the link on Blackboard

nn.Linear() function

Group Exercise#1

Create a new Linear layer with the following structure:

> The first layer has 2 input nodes and 16 output nodes.

```
[ ] # your code here
    # ...
```

Group Exercise#2

> Apply a tensor through your linear layer now.

> Change the value in torch.manual_seed(0) to something else, generate new inputs, and pass the tensor through your linear layer again.

> Observe the the output values.

```
[ ] # your code here.
    # ...
```

Activation Functions: nn.Sigmoid() nn.ReLU() etc

Group Exercise#3

Experiment with different activation functions like sigmoid, tanh, and relu, and then pass a tensor through the linear layer you created for Group Exercises #1 and #2.

Change the value in torch.manual_seed(2) to something else, generate new inputs, and pass the tensor through your linear layer again.

Take a look at the output values and make sure they match what you were expecting!

Combining everything to make an MLP

Group Exercise#4

Let's create three Linear layers and connect them in sequence to build an MLP with the following structure:

> The first layer has 2 input nodes and 3 output nodes.

> The second layer takes 3 input nodes and outputs 6 nodes.

> The final layer connects 6 input nodes to 2 output nodes.

```
[ ] # your code here
    # ...
```

Group Exercise#5

> Apply a tensor through your MLP now.

```
[ ] # your code here
    # ...
```

Reza Demo

Today's Agenda

- Simple Multilayer Perceptrons (MLP) Implementation using PyTorch
  - Basic functions and utilities

- Modular MLP Implementation using PyTorch
  - structural aspect
  - following the conventions of the research community

Modular Code Multilayer Perceptron using MLP

A multilayer perceptron is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers. Create a network class with two methods:

- *init()*
- *forward()*

```python
import torch
from torch import nn

# You can give any name to your new network, e.g., SimpleMLP.
# However, you have to mandatorily inherit from nn.Module to
# create your own network class. That way, you can access a lot of
# useful methods and attributes from the parent class nn.Module

class SimpleMLP(nn.Module):
  def __init__(self):
    super().__init__()
    # your network layer construction should take place here
    # ...
    # ...

  def forward(self, x):
    # your code for MLP forward pass should take place here
    # ...
    # ...
    return x
```

- Let's jump into the notebook for a detailed discussion
  - https://github.com/alimoorreza/CS167-sp26-notes/blob/main/Day21_Building_Modular_MLP_with_PyTorch.ipynb

Open the link on Blackboard

Reza Demo