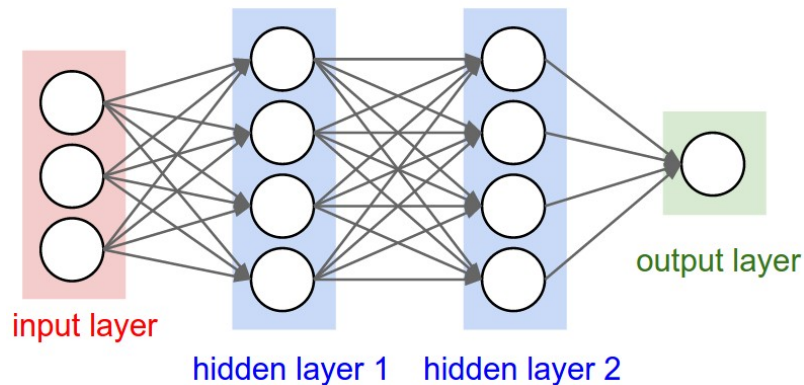


# Neural Network Example

Xây dựng neural network với hai tầng ẩn (hidden layer). Các trọng số và mối liên hệ giữa các tầng ẩn được thiết lập và cài đặt bằng việc sử dụng API của thư viện PyTorch

## Neural Network Overview



## MNIST Dataset Overview

```
In [ ]: from __future__ import absolute_import, division, print_function

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.autograd import Variable
import numpy as np
```

```
In [ ]: # Chuẩn bị dữ liệu
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Chuyển đổi sang định dạng float32.
x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)
# Chuẩn hóa ảnh từ from [0, 255] to [0, 1].
x_train, x_test = x_train / 255, x_test / 255
x_train, x_test, y_train, y_test = torch.from_numpy(x_train), torch.from_numpy(x_test), torch.from_numpy(y_train), torch.from_numpy(y_test)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11493376/11490434 [=====] - 0s 0us/step
11501568/11490434 [=====] - 0s 0us/step
```

```
In [ ]: # MNIST dataset parameters.
num_classes = 10 # total classes (0-9 digits).
num_features = 784 # data features (img shape: 28*28).

# Network parameters.
n_hidden_1 = 128 # 1st layer number of neurons.
# n_hidden_2 = 256 # 2nd layer number of neurons.
```

```
In [ ]: batch_size = 16
```

```
In [ ]: trainloader = []
for (i,j) in zip(x_train, y_train):
    trainloader.append([i,j])
trainloader = torch.utils.data.DataLoader(trainloader, shuffle=True, batch_

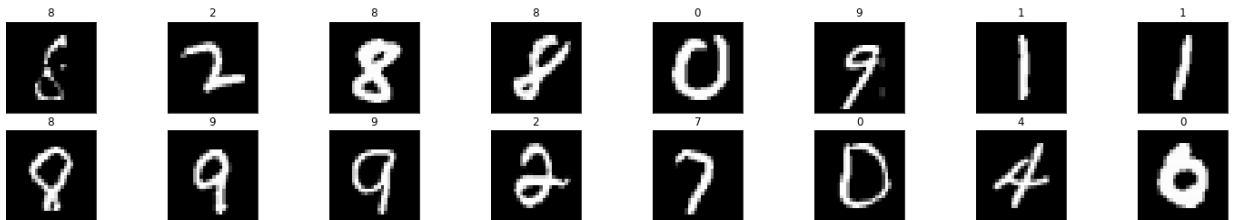
testloader = []
for (i,j) in zip(None, None):
    testloader.append([i,j])
testloader = torch.utils.data.DataLoader(None, shuffle=True, batch_size=Non
```

## Hiển thị một vài ví dụ

```
In [ ]: import matplotlib.pyplot as plt
%matplotlib inline

dataiter = iter(trainloader)
images, labels = dataiter.next()
images = images.numpy()

# hiển thị dữ liệu theo từng batch và nhãn tương ứng
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(batch_size):
    ax = fig.add_subplot(2, batch_size/2, idx+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(images[idx]), cmap='gray')
    ax.set_title(str(labels[idx].item()))
```



```
In [ ]: import torch.nn as nn
import torch.nn.functional as F

## định nghĩa mạng
class MLPModel(nn.Module):
    def __init__(self):
        super(MLPModel, self).__init__()
        self.fc1 = nn.Linear(num_features, None)
        # linear layer (n_hidden -> hidden_2)
        self.fc2 = nn.Linear(None, None)

    def forward(self, x):
        # chuyển từ định dạng ma trận thành vector
        x = x.view(-1, 28 * 28)
        # add hidden layer, with relu activation function
        x = F.relu(self.fc1(None))
        x = F.relu(self.fc2(None))
        return x

# initialize the NN
model = MLPModel()
print(model)
```

```
In [ ]: import torch.optim as optim
# trong hàm loss cross entropy đã áp dụng hàm soft max cho vector đầu ra
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

```
In [ ]: def accuracy():
    correct = 0
    total = 0
    # since we're not training, we don't need to calculate the gradients for
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            # calculate outputs by running images through the network
            outputs = model(None)
            # the class with the highest energy is what we choose as prediction
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return correct/total
```

```

In [ ]: losses = []
        for epoch in range(10): # loop over the dataset multiple times

            running_loss = 0.0
            for i, data in enumerate(trainloader, 0):
                # get the inputs; data is a list of [inputs, labels]
                inputs, labels = None

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward + backward + optimize
                outputs = model(None)
                loss = criterion(None, None)
                loss.backward()
                optimizer.step()

                # print statistics
                running_loss += loss.item()
            losses.append(running_loss)
            if epoch % 10 == 9:
                print('Iteration: %d, accuracy: %.3f' % (epoch + 1, accuracy()))

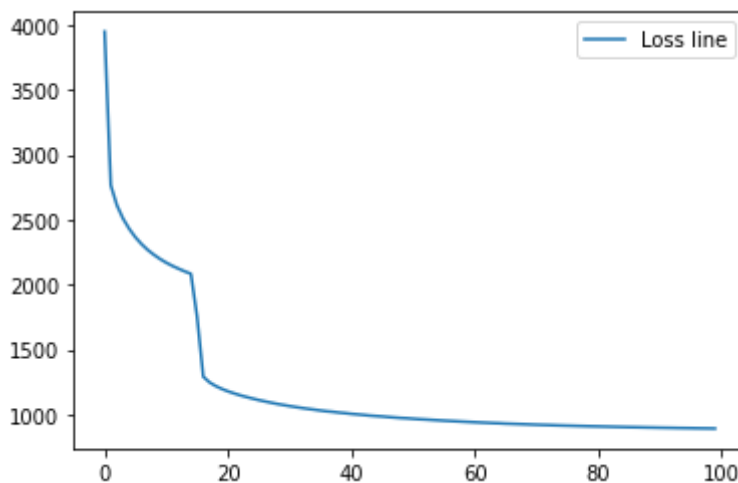
        print('Finished Training')

```

```

In [ ]: # Biểu đồ biểu diễn độ biến thiên của hàm mất mát qua các vòng lặp
        plt.plot([i for i in range(len(losses))], losses, label='Loss line')
        # plt.plot(X, np.array(W * X + b), label='Fitted line')
        plt.legend()
        plt.show()

```



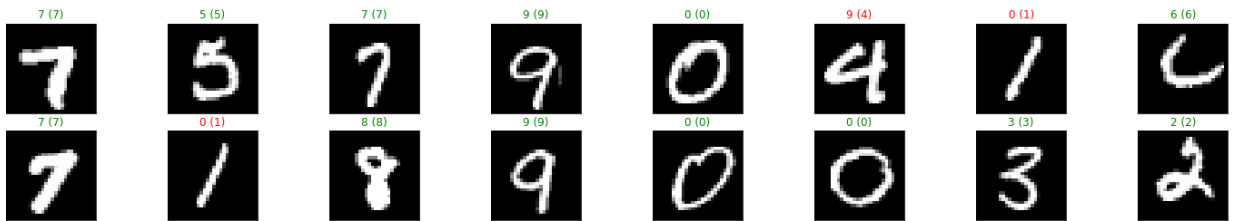
## Bài tập

- Tiến hành huấn luyện mô hình với số lượng epoch lần lượt là 50, 100, 150. Rút ra nhận xét.
- Hiển thị một số kết quả

```
In [ ]: # obtain one batch of test images
dataiter = iter(testloader)
images, labels = dataiter.next()

# get sample outputs
output = model(images)
# convert output probabilities to predicted class
_, preds = torch.max(output, 1)
# prep images for display
images = images.numpy()

# plot the images in the batch, along with predicted and true labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(batch_size):
    ax = fig.add_subplot(2, batch_size/2, idx+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(images[idx]), cmap='gray')
    ax.set_title("{} ({}).format(str(preds[idx].item()), str(labels[idx].item()),
                        color=("green" if preds[idx]==labels[idx] else "red"))
```



In [ ]:

## Bài tập

- Thiết lập thêm 3 mạng nơ ron với số tầng ẩn lần lượt là 3,5 (số lượng nơ ron của các tầng tùy ý). Với mỗi mạng mới được sinh ra, áp dụng learning-rate lần lượt bằng 0.1, 0.05, 0.001 để huấn luyện. Huấn luyện qua khoảng 100 epoch.
- Chia tập train hiện có thành 2 phần. **85%** dữ liệu làm tập **train**, **15%** còn lại làm tập **dev**. Huấn luyện các mô hình tạo ra ở bước trên với tập **train** mới, sau đó kiểm thử trên tập **dev**.
- Dùng các model kiểm thử trên tập **test**. So sánh model cho kết quả tốt nhất khi kiểm thử trên 2 tập này và rút ra nhận xét về vai trò của tập **dev**

In [ ]: