CS167: Machine Learning

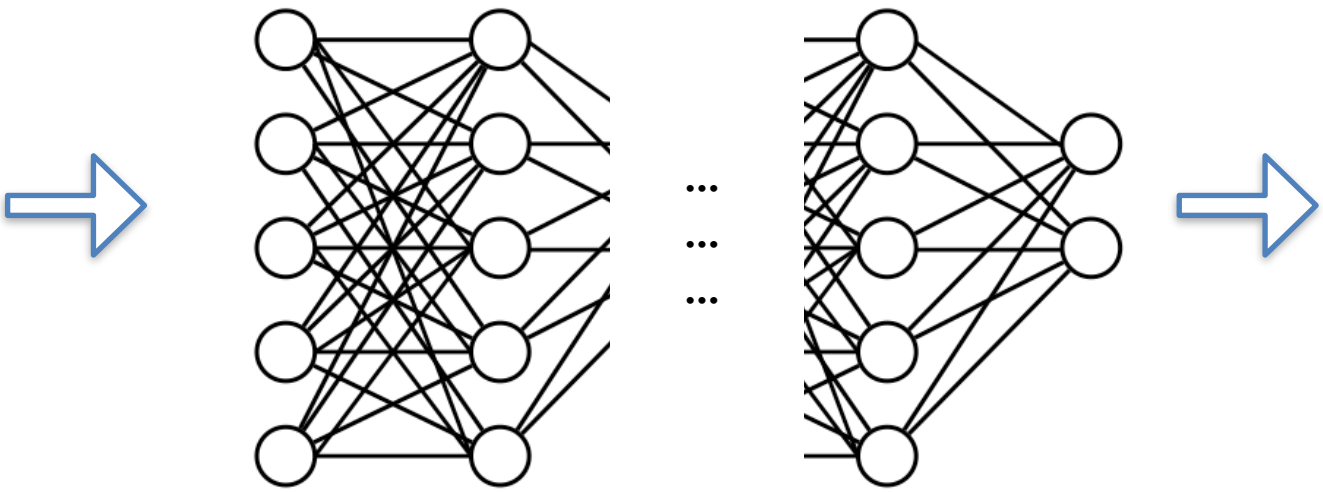
Modular Implementation of Multilayer Perceptron (MLP) with PyTorch

Thursday, April 11th, 2024

Drake UNIVERSITY

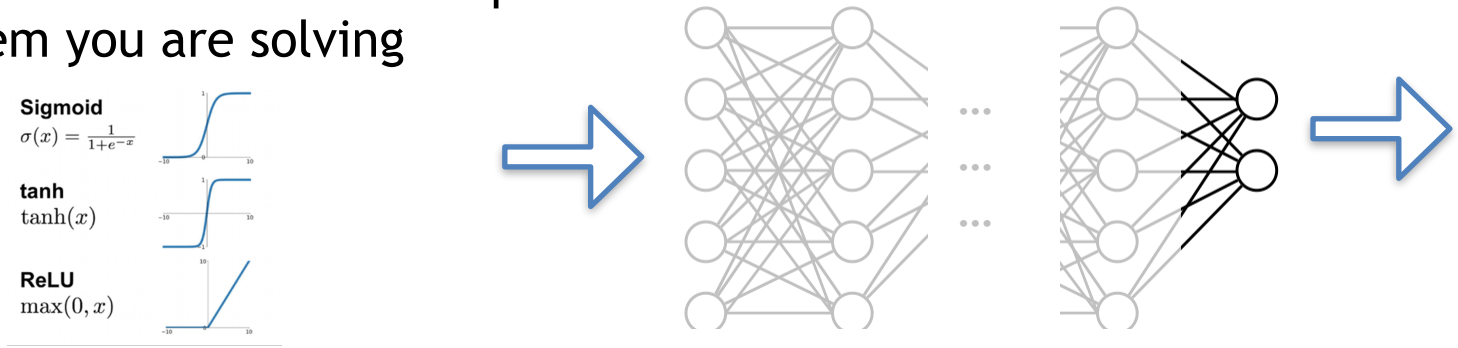
Recap: MLP (Network) Structure

- Each of these questions need to be answered before you set up your neural network:
  - how many hidden layers should I have? (depth)
  - how many neurons should be in each layer? (width)
  - what should your activation be at each of the layers?



Recap: Final Output Nodes

- In general, the complexity of your network should match the complexity of your problem. The final output nodes should be related to what kind of problem you are solving



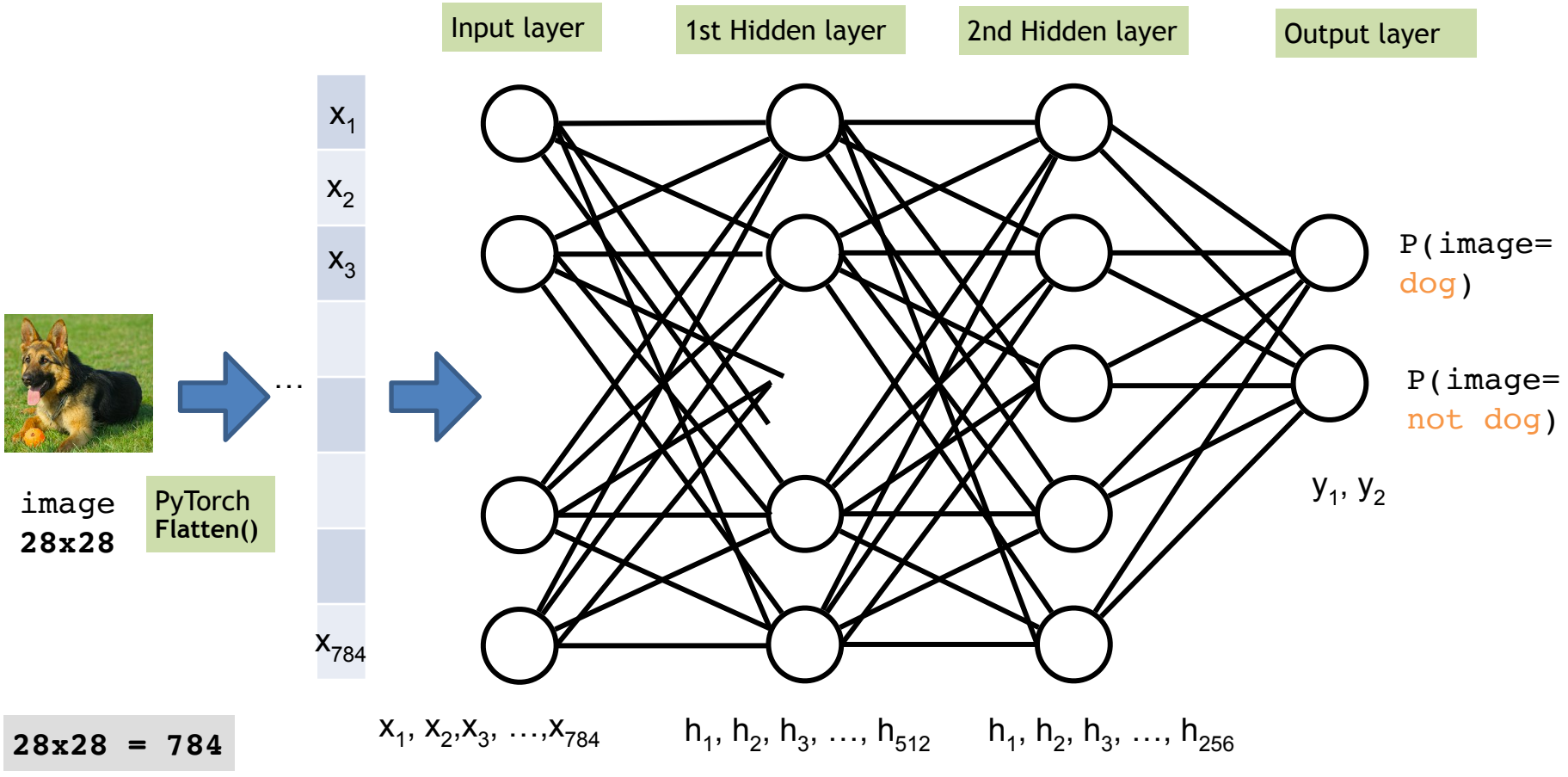
Activation Function	Function	Lower bound	Upper bound	Type of Machine Learning
Linear	$f(z) = az$	$-\infty$	∞	regression where results can be negative
Rectified Linear Unit (ReLU)	$relu(z) = max(0, z)$	0	∞	regression where results can't be negative
Sigmoid	$sigmoid(z) = \frac{1}{1+e^{-z}}$	0	1	binary classification
Softmax	$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$	0	1	multiclass classification

Today's Agenda

- Simple Multilayer Perceptrons (MLP) Implementation using PyTorch

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



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()*

```
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
```

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()
- nn.CrossEntropyLoss()
- torch.optim.SGD

- Let's jump into the notebook for a detailed discussion.