CS4910: Deep Learning for Robotics

David Klee
klee.d@northeastern.edu

T/F, 3:25-5:05pm
Behrakis Room 204

https://www.ccs.neu.edu/home/dmklee/cs4910_s22/index.html

https://piazza.com/northeastern/spring2022/cs4910a/home

Neural Networks in Pytorch

Today’s Agenda

1. Discuss Neural Network Modules
2. How to Train Your Network
3. Example: design *nn.Module*
4. Overview of nuro arm API

Installing Pytorch [latest download info]

Mac

```
conda install pytorch torchvision torchaudio -c pytorch
```

Windows

```
conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

Linux

```
conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

*make sure to do this in your (cs4910) environment

A tensor is a multi-dimensional array (similar to numpy.ndarray)

t = torch.tensor(data, dtype, device, requires_grad)

data -> ArrayLike object, ideally numpy array

dtype -> optional, often use `torch.float32` for networks

device -> optional, default=torch.device(“cpu”)

requires_grad -> optional, default = False

Basic commands on Tensors

t.size() or t.shape -> get shape of tensor

t.to(dtype, device) -> send tensor to new device, or change its datatype

t.view(new_shape) -> returns view of the tensor with new shape (like numpy.reshape)

t.squeeze() -> removes any dimensions with size 1

t.unsqueeze(i) -> inserts dimension with size 1 at i^{th} dimension

t.expand(*sizes) -> returns view with dimensions repeated according to \`sizes\`

t.detach() -> returns new tensor without gradient information

t.numpy() -> returns numpy array with tensor's data

What is a neural network?

$$y \approx f(x, \theta)$$

y : desired output of network

f: non-linear function whose derivative exists

θ: weights that parametrize function *f*

x: inputs (domain) of the neural network

nn.Linear(in_features: int, out_features: int, bias: bool)

$$y = xA^T + b$$

$$x.\texttt{size()} = (*, H_{in})$$

$$y.\texttt{size()} = (*, H_{out})$$

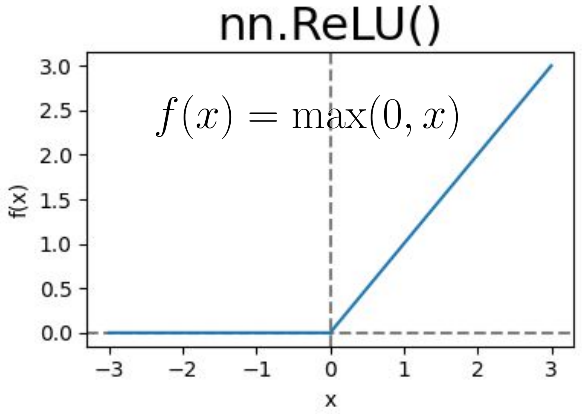
$$A.\texttt{size()} = (H_{out}, H_{in})$$

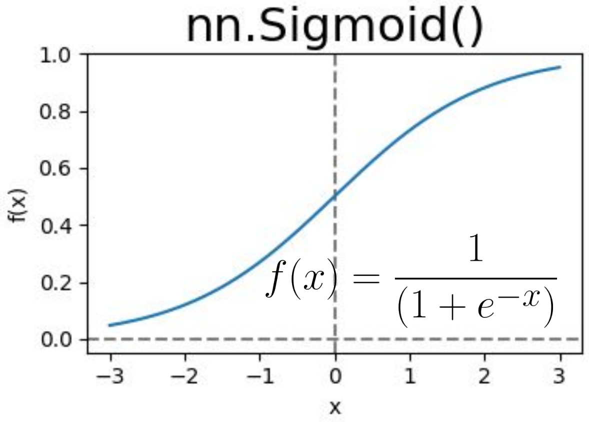
$$b.\texttt{size()} = (H_{out})$$

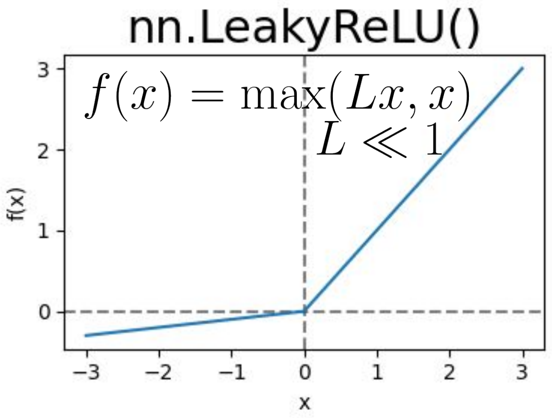
Use cases...

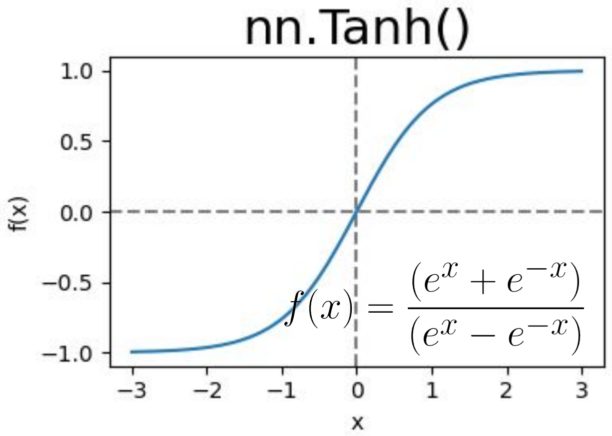
- Unstructured data
- All data points are useful to each other
-

Non-Linearity Functions (Activation Functions)

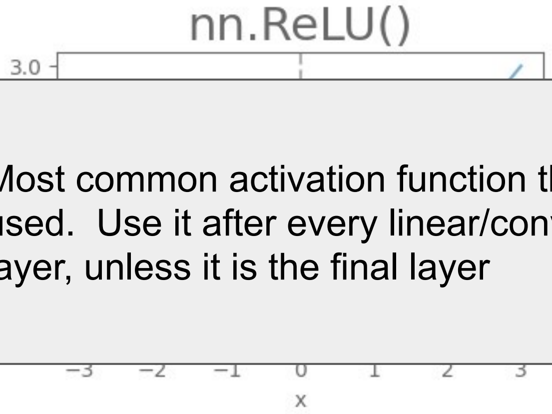




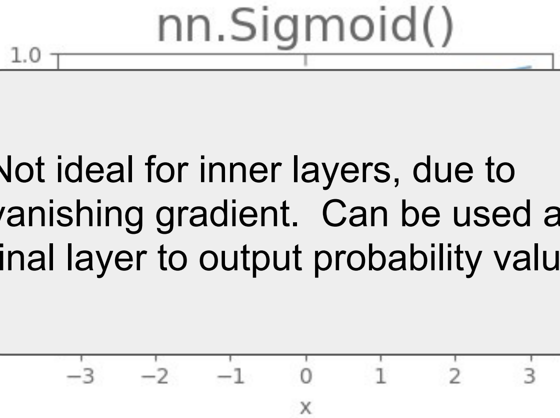




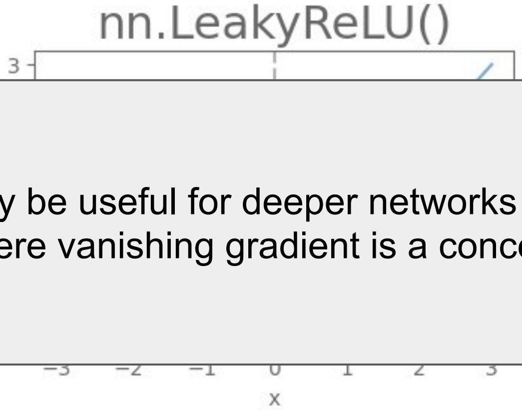
Use Cases for Activation Functions



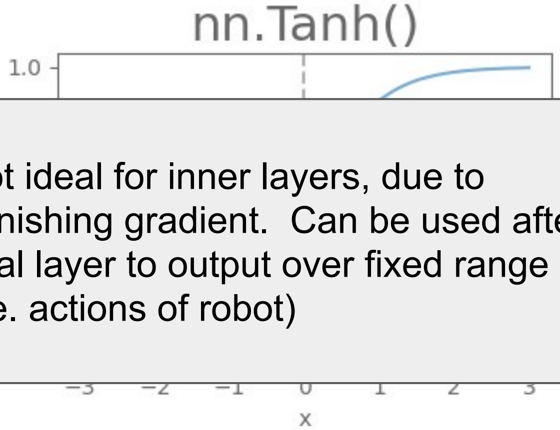
Most common activation function that is used. Use it after every linear/conv layer, unless it is the final layer



Not ideal for inner layers, due to vanishing gradient. Can be used after final layer to output probability value



May be useful for deeper networks where vanishing gradient is a concern



Not ideal for inner layers, due to vanishing gradient. Can be used after final layer to output over fixed range (i.e. actions of robot)

Example 1: Multi Layer Perceptron (MLP)

Task: Design nn.Module that predicts probability of grasp success given objects pose

Input: object pose (6D) and action position (x, y, th)

Output: probability of success

Guidance: use 3 linear layers with 128 hidden units

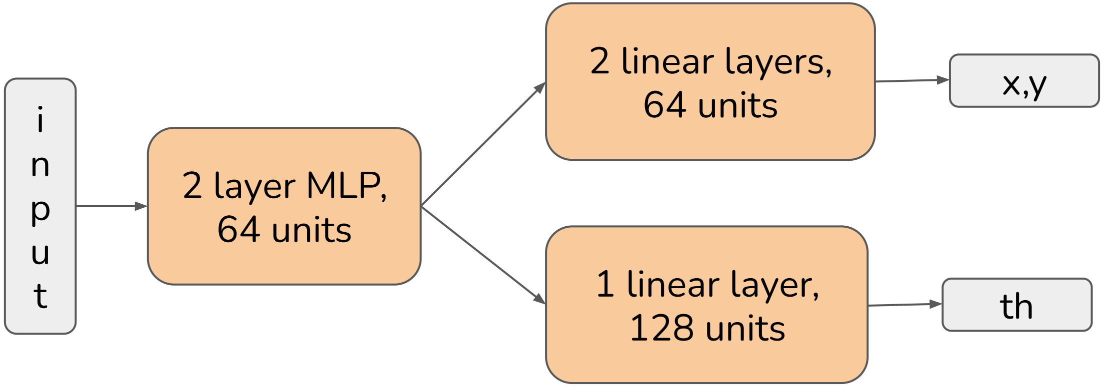
Example 2: Multi Layer Perceptron (MLP)

Task: Design nn.Module that predicts grasp action

Input: object pose (6D)

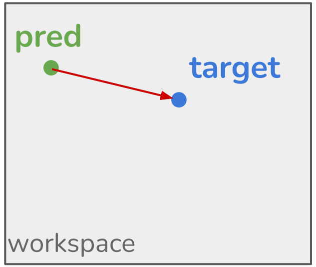
Output: action position (x, y, th)

Guidance: use multi-head architecture to predict (x,y) separately from angle



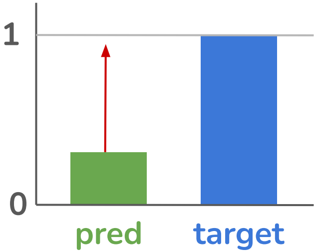
Common loss function

nn.MSELoss() <- Mean Squared Error, useful for regression task



$$\mathcal{L}_{mse}(\tilde{y}, y) = \frac{1}{N}\sum_{i}^{N} \|\tilde{y} - y\|^2$$

nn.BCELoss() <- Binary Cross Entropy, useful for classification



$$\mathcal{L}_{bce}(\tilde{y}, y) = -\frac{1}{N}\sum_{i}^{N} \Big(y \log(\tilde{y}) + (1 - y)\log(1 - \tilde{y})\Big)$$

Let's create loss functions for our two examples:

Optimizing the network through backpropagation

0. Create optimizer

```
optimizer = torch.optim.SGD(network.parameters(), lr=learning_rate)

for epoch_id in range(n_epochs):
    loss = compute_loss(network)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

1. Compute Loss
2. Zero the gradients
3. Perform back propagation
4. Step weights along gradient

Common optimizers are torch.optim.SGD, torch.optim.Adam, and torch.optim.RMSprop. There should only be minor variations in performance. Learning rate can vary, but 1e-3 is a good first guess

Next class...

- HW1 review?
- Convolutional Layers
- Normalization Layers
- DataLoaders and Data augmentation
- Debugging the training process
- Using GPU's in the cloud

Using xArm with nuro.arm API

https://dmklee.github.io/nuro-arm/

$ git clone https://github.com/dmklee/nuro-arm.git

$ cd nuro-arm

$ pip install .

Calibrating xArm

Make sure it is plugged in and connected to computer

```
$ python nuro_arm/robot/calibrate.py
```

Programming arm

```
$ python nuro_arm/examples/record_movements.py
```

Survey to provide feedback

https://forms.gle/1heiVgXEEZ6abWhX9