HUMAN-ROBOT INTERACTION

TUTORIAL 2: Robot learning

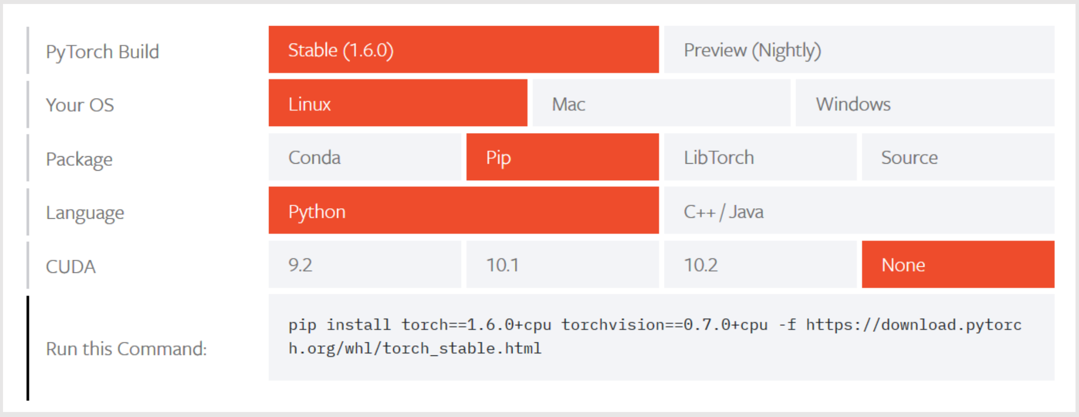
Author: Prof. Pablo Lanillos | Department of Artificial Intelligence | Radboud University

Goal: The goal of this tutorial is to incorporate robot learning techniques in HRI applications.

1. Neural networks reminder (Part 1)

Open the jupyter notebook *part1/HRI_Tutorial2_part1_intro2FFN.ipynb* or the (part1/*HRI_Tutorial2_part1_intro2FFN.html* for the slides) and explore and run the examples. There is an example of a Feed-forward network (FFN), programmed from scratch and a FNN programmed with the Pytorch environment. You can also watch the slides in *HTML*.

You have to install PyTorch for CPU (CUDA none) or if you have GPU you can install it with the cuda version that you have on your system). Find the installation command there: https://pytorch.org/get-started/locally/

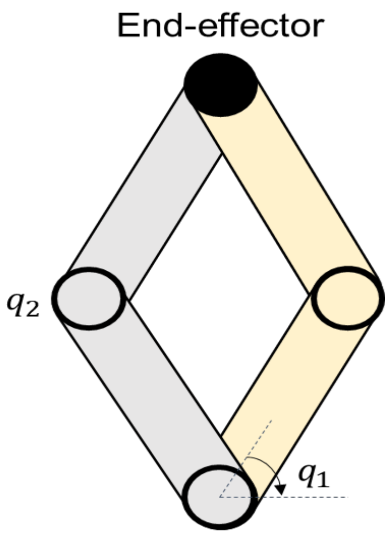


PyTorch Build	Stable (1.6.0)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
CUDA	9.2	10.1	10.2	None

Run this Command:

```
pip install torch==1.6.0+cpu torchvision==0.7.0+cpu -f https://download.pytorc
h.org/whl/torch_stable.html
```

2. Mixed Density Networks (Part 2)



Create your own jupyter notebook and implement a MDN in PyTorch following the tutorial that you can find in *HRI_Tutorial2_part2_MDN.html*. It is highly recommendable to read Bishop paper on the MDN[1]. Here, you learn two important aspects: 1) merge connectionist and Bayesian approaches and 2) how to solve the inverse model problem when two different inputs have the same output. The figure on the left shows this problem using a robotic arm. Where two different joint configurations yield to the same position of the end-effector. This is, two solutions of the joint angles q have the same location in the 2D space. One way to solve this is to use the combination of probability distributions (e.g., Multivariate Gaussians) with techniques like Gaussian Process Regression (GPR) or the use of Probabilistic neural networks as the MDN.

[1] Christopher M. Bishop. Mixed Density Networks (1994). https://publications.aston.ac.uk/id/eprint/373/1/NCRG_94_004.pdf

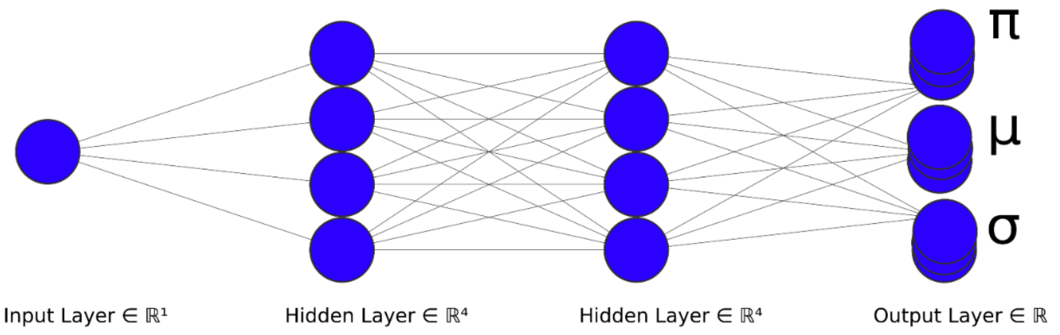
DATA GENERATION: use the following function to generate the data:

```
noise = np.random.normal(size=(n_samples))
y_data = np.random.uniform(-10.5, 10.5, n_samples)
x_data = 7*np.sin(0.75*y_data) + 0.5*y_data + noise
```

The MDN objective is to learn the probability distribution of the output *y* given the input *x*: $p(y|x)$. We model this by a Mixture of Gaussians:

$$p(y|x) = \sum_{i=0}^{K} \pi_i(x)\, \mathcal{N}(y,\ \mu_i(x),\ \sigma_i(x))$$

Where K is the number of Gaussians used to model the distribution. To compute the coefficients π_i, the means μ_i and the variances σ_i we will use a neural network as follows:



When there are more inputs and more outputs we need to create the needed nodes for each of the components. Here, to simplify we are going to assume that the output components are independent so we can use a diagonal covariance matrix. This simplifies considerably the computation of the multivariate Gaussian. We have two other restrictions. The coefficient must sum to 1 and the variances must be positive. For two inputs and two outputs, the architecture will look as follows:

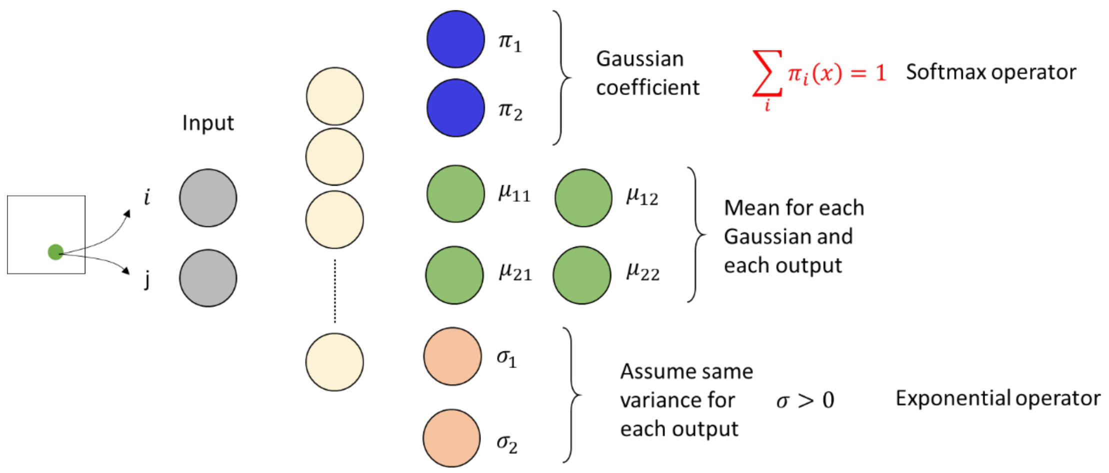


Figure 1: Network architecture. Node connections have been omitted for clarity. The location of the ball is passed through the network to predict the joint angles needed. The coefficients output use a softmax operator to force it to sum to 1. The variance is forced to be positive by an exponential operator. We need one mean per output and per Gaussian.

The softmax operator is as follows: $\pi_i = \frac{e^{z_i}}{\sum_i e^{z_i}}$, where z_i is the output of the neuron that corresponds to the coefficient. The same applies to the exponential operator: $\sigma_i = e^{z_j}$, where z_i is the output of the neuron that corresponds to the variance. The mean does not need any modification.

All layers of the network are linear. For the hidden layer, we are going to use the *tanh* activation function to capture the non-linear nature. From the input to the hidden layer is like in a normal FFN for regression and from the hidden layer to the output, we are going to split the network into 3 blocks as in the :

$$z_\pi \leftarrow Linear(n_{hidden}, n_{gaussians})$$

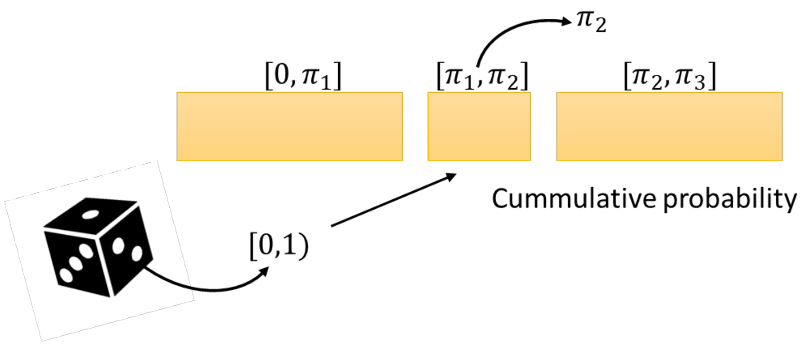
$$z_\sigma \leftarrow Linear(n_{hidden}, n_{gaussians})$$

$$z_\mu \leftarrow Linear(n_{hidden}, n_{gaussians} * n_{output})$$

TRAINING: In order to train the network we use a common loss function called the negative log-likelihood:

$$loss = -\log \sum_{i=0}^{K} \pi_i(x)\, \boldsymbol{\mathcal{N}}(y,\ \mu_i(x),\ \sigma_i(x))$$

PREDICTION: Instead of only using the forward pass to compute the prediction. We need to sample from the output Mixture of Gaussians. First, we select the kernel (Gaussian) using the coefficients. As we know that they sum to 1, we can throw a random number between 0 and 1 and then incrementally sum the coefficients.



Once the Gaussian has been selected then we just sample from the Normal distribution

3. Robot learning

Develop a controller to follow the ball with the hand. We are going to use only the 2 Degrees of Freedom (DOFs) of the left shoulder. The rest of the joints will not change. We want to learn the inverse model: the mapping between the sensory information (location of the ball in the image) and the body configuration (joint angles). This mapping will be learnt with a neural network.

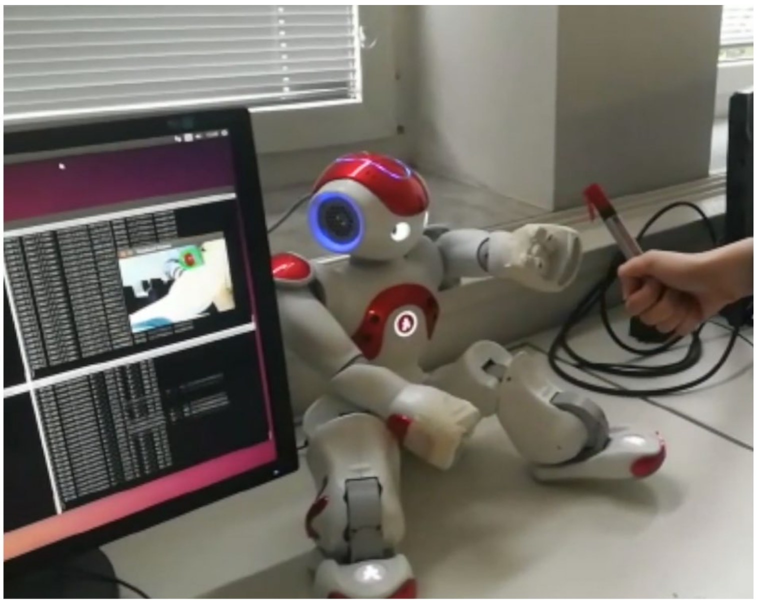


Figure 2. Example of the final controller in the real NAO. The robot moves the arm to follow the red pen as the user is moving it. To get the position of the pen colour segmentation has been performed. Note that the head is slightly moved towards the left arm to have a better field of view.

3.1 Develop a program to generate the needed data. We are going to simulate that we move the arm of the robot towards the ball and then you can use the keyboard to store values of the joints and the ball location in the image.
3.2 Learn the mapping using a FFN and use the network prediction to move the arm towards the ball.

Specification of the FFN:

- Input layer: Input with 2 neutrons (i,j) coordinates of the ball in the image.
- Hidden layer: 1 hidden layer of 6 neurons
- Output layer: Output of 2 neurons (shoulder pitch and roll angles).

Activation function: *tanh* for the hidden layer and linear for the output layer (regression)

3.3. Learn the mapping using a MDN and use the network prediction to move the arm towards the ball. As the MDN has a probabilistic output you have to sample the prediction from the multivariate Gaussian.

Questions

1. Are we getting the same result as when using the FFN? Why?
2. What is the relation of the mean output of the MDN and the output of a normal FFN?
3. How we can scale this problem to use images instead of the segmented location of the ball?