Machine Learning for Communications
Final Project

January 17, 2020

Institute for Communications Engineering
Technical University of Munich
Prof. Dr. sc. techn. Gerhard Kramer

The following lecture notes are part of the course "Machine Learning for Communications" offered by the Institute for Communications Engineering at the Technical University of Munich. All content is subject to copyright restrictions. If you are planning to use any of the material, please contact Prof. Dr. sc. techn. Gerhard Kramer (gerhard.kramer@tum.de).

Guidelines

- Please hand in the project in a group of at most **5 students**.
- The solutions to the following tasks should be sent as **one zip file** to `mlcomm@lnt.ei.tum.de` by **Mar 01, 2020 23:59 CET**. The naming scheme of the zip file must be `mlcomm_project_NAME1_NAME2_NAME3_NAME4_NAME5.zip`, where `NAMEX` should be replaced by the family name of each contributor. Additionally, include a file `names.txt` in your submission, which lists the names of all contributors and their matriculation numbers – one name per line and the matriculation number separated by a semicolon. If the zip file is too large, a Dropbox/Box/LRZ Sync&Share link in the email is fine as well.
- The project will be graded as either "passed" or "failed". If it is passed a bonus of 0.3 will be given to your exam grade. However, no bonus will be given if your exam grade is worse than 4.0.
- Grading: to pass the project, you need to have reasonably correct attempts at solving **two out of the three tasks**. If we find that significant parts of your solutions are copied from elsewhere, it will be graded as "failed".
- The code for each of three tasks should go into individual files named
  - `task1_training.py`,
  - `task1_testing.py`,
  - `task2.py`,
  - `task3.py`.

  Alternatively, Jupyter notebooks can be used as well, but should follow the same naming scheme.
- The set of allowed libraries is limited to: Python standard library, matplotlib, numpy, scipy, torch, torchvision.
- We can provide limited access to GPUs. If you need one, drop an email to `lars.palzer@tum.de`.

1 Image Classification with Neural Networks

During the PyTorch introductions in the tutorials, we used the MNIST dataset of handwritten digits as a typical example for classification tasks. This time, we are investigating the Kaggle Cats and Dogs dataset (by clicking on "Leaderboard", you can get an idea of how well your model works).

- The Cats and Dogs dataset consists of images where each image belongs to exactly one of the two categories "cats" and "dogs".

- We use a **filtered version of the Kaggle data set**. The dataset can be downloaded from `https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip` (68.8 MB). After downloading, you can load, e.g., the training data set via

```
torchvision.datasets.ImageFolder(root=PATH,
                        transform = torchvision.transforms.ToTensor())
```

- It is up to you to decide which neural network architecture you want to use (both conventional and convolutional networks) are allowed. Also the number of hidden layers, number of neurons, filter sizes, etc. is up to you.

- For an introduction to convolutional neural networks, you may start with the following resources:
  - Video recording of last year's introduction (Moodle → Video Recordings WS 2018/2019 → `Machine_Learning_for_Communication_20181218`
  - Stanford course notes on convolutional nets (click)
  - Corresponding lecture on convolutional nets (click)
  - Here is an introduction into how to use convolutional networks in PyTorch and how to use the GPU for training (at the bottom).

- For this task, the training and test part should be split into two files `task1_training.py` and `task1_testing.py` (or corresponding notebooks). In `task1_training.py`, the training of the neural network should be performed using only the images from the `train` folder, while `task1_testing.py` should test the trained model on the `validation` folder and print out the achieved accuracy. You can use `torch.save` for this purpose (see here for an example). Please also provide the trained model in the zip file so that it can directly be loaded for testing.

2 Markov-Chain Monte Carlo for Log-Likelihood Calculation

Consider a point-to-point MIMO transceiver with N_t transmit and N_r receive antennas with the channel model

$$\underline{y}_i = \underline{\underline{H}}\,\underline{x}_i + \underline{n}_i, \quad i = 1, \ldots, n. \tag{1}$$

The transmit vector $\underline{x} = (x_1, x_2, \ldots, x_{N_\mathrm{t}})^\mathrm{T}$ entries $x_j, j = 1, 2, \ldots, N_\mathrm{T}$ are taken from an M-ary QAM signal constellation. We have an average power constraint $\mathrm{E}\left[\underline{X}^\mathrm{H}\underline{X}\right] = 1$. The noise $\underline{N}$ is zero-mean, multivariate, circularly-symmetric Gaussian with a scaled identity as covariance , i.e., $\underline{N} \sim \mathcal{N}(0, \sigma^2\underline{\underline{I}})$. We consider a flat-fading channel model and the entries of $\underline{\underline{H}}$ are iid. circularly-symmetric Gaussian with zero mean and unit variance. The channel is known to both the transmitter and receiver.

We associate a length mN_t bit binary vector with each $\underline{x} \in \mathcal{X}^{N_\mathrm{t}}$ using the mapping $\chi : \mathcal{X}^{N_\mathrm{t}} \to \{0,1\}^{mN_\mathrm{t}}$:

$$\chi(\underline{x}) = \underline{b} = (b_1, b_2, \ldots, b_{mN_\mathrm{t}}). \tag{2}$$

Practical systems usually use binary forward-error correction (FEC) with a soft-input decoding algorithm. In this case, the input to the decoder is a vector of log-likelihood ratios (LLRs) with entries

$$L_i = \log\left(\frac{P_{B_i|\underline{Y}}(0|\underline{y})}{P_{B_i|\underline{Y}}(1|\underline{y})}\right) = \log\left(\frac{\sum_{\underline{x}\in\mathcal{X}_{i,0}^{N_\mathrm{t}}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}{\sum_{\underline{x}\in\mathcal{X}_{i,1}^{N_\mathrm{t}}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}\right), \quad i = 1,\ldots,mN_\mathrm{t} \tag{3}$$

where we assumed that the constellation symbols are equally likely and the set $\mathcal{X}_{i,b}^{N_\mathrm{t}}$ contains all vectors in $\mathcal{X}^{N_\mathrm{t}}$ which have a value of b in the i-th bit position, i.e., $\mathcal{X}_{i,b}^{N_\mathrm{t}} = \left\{\underline{x} \in \mathcal{X}^{N_\mathrm{t}} : [\chi(\underline{x})]_i = b\right\}$.

Calculating the log-likelihood values becomes infeasible when the number of transmit antennas N_t is too large. For example, for 64-QAM and $N_\mathrm{t} = 8$ antennas we have $\left|\mathcal{X}^{N_\mathrm{t}}\right| = 64^8 = 2^{48} \approx 10^{14}$. We circumvent this complexity by calculating the LLRs as

$$L_i = \log\left(\frac{\sum_{\underline{x}\in\tilde{\mathcal{X}}_{i,0}^{N_\mathrm{t}}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}{\sum_{\underline{x}\in\tilde{\mathcal{X}}_{i,1}^{N_\mathrm{t}}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}\right), \quad i = 1,\ldots,mN_\mathrm{t} \tag{4}$$

where the respective sets $\tilde{\mathcal{X}}_{i,b}^{N_\mathrm{t}}, b \in \{0,1\}$ have a much smaller cardinality. However, the question arises how to find these these sets. We will use Markov-Chain Monte Carlo (MCMC) sampling to find samples $\underline{b}$ from $P_{\underline{B}|\underline{Y}}(\cdot|\underline{y})$ and store them in $\tilde{\mathcal{X}}_{i,b}^{N_\mathrm{t}}, b \in \{0,1\}$ accordingly. The idea is also described in Sec. III-A of

[1] R. R. Chen, R. Peng, A. Ashikhmin, and B. Farhang-Boroujeny, "Approaching MIMO capacity using bitwise Markov Chain Monte Carlo detection," IEEE Trans. Commun., vol. 58, no. 2, pp. 423–428, Feb. 2010.

- Implement the MCMC sampling approach of [1] for a MIMO transceiver with $N_\mathrm{t} = N_\mathrm{r} = 2$ and $N_\mathrm{t} = N_\mathrm{r} = 8$ and 16-QAM. *Hint: You can omit the λ_i in the paper above, as we do not consider iterative demapping, i.e., we do not have LLRs coming from a previous run of the channel decoder.*

- Calculate the BICM capacity $\sum_{i=1}^{N_\mathrm{t}m} \mathrm{I}(B_i;\underline{Y}) = \sum_{i=1}^{N_\mathrm{t}m} \mathrm{I}(B_i;L_i)$ for the above scenarios for the signal-to-noise ratio (SNR) regime of 0 dB to 15 dB. We define the SNR as

  $$\mathrm{SNR} = \frac{\mathrm{E}\left[\underline{X}^\mathrm{H}\underline{X}\right]}{\sigma^2} = \frac{1}{\sigma^2}.$$

  We provide a function `bicm_cap_mc(B, L)` that calculates the BICM capacity via a Monte Carlo approximation. Here, the parameters `B` and `L` are matrices of dimension $mN_\mathrm{t} \times n_\mathrm{s}$, where n_s is the number of samples. The return variable `R` denotes the achievable rate. You may find the functions in `mlcomm_project.py` helpful to deal with the construction of QAM constellations, Gray labels, etc.

- Plot the achievable rate vs. SNR in dB for both scenarios. Also, compare to the exact solution (i.e., with the LLRs calculated as in (3)) for $N_\mathrm{t} = N_\mathrm{r} = 2$.

3 Expectation Maximization

You are given a dataset `EM_data.npy` (load with `np.load`) of noisy receive data points $y_i \in \mathbb{C}, i = 1, \ldots, 10\,000$ after transmission over an optical channel. The transmission over the optical fiber is modeled as

$$Y = \Delta X + N$$

where $\Delta \in \mathbb{C}$ and X is from a M-QAM constellation $\mathcal{X}$ of unknown size M. The noise N is zero mean Gaussian with unknown variance σ^2. In this task, you should do the following:

- Use the expectation maximization (EM) algorithm to determine the unknown model parameters σ^2, Δ and the distribution P_X on the constellation symbols. Print those results after performing this estimation.
- *Hint 1:* A scatter plot of the transmit and receive data is very helpful to gain first insights.
- *Hint 2:* Think carefully about how to initialize the algorithm. As pointed out in the lecture, an initialization with K-Means may be beneficial to get a good initial starting point for EM.