

# Machine Learning for Communications

## Final Project

February 17, 2018

---

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](mailto: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@int.ei.tum.de` by **Feb 25, 2019 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.
- As announced during class, 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.
- 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, numpy, scipy, tensorflow.
- We can provide limited access to GPUs. If you need one, drop an email to `fabian.steiner@tum.de`.

## 1 Image Classification with Neural Networks

During the Tensorflow introductions in the tutorials, we used the MNIST dataset of handwritten numbers as a typical example for classification tasks. This time, we are investigating the CIFAR-10 dataset.

- The CIFAR-10 dataset consists of images where each image belongs to exactly one category. These categories are “airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship”, “truck”.
- The dataset can be downloaded from <https://www.cs.toronto.edu/~kriz/cifar.html>. The website also describes how the images are stored.

- 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 this task, the training and test part should be split in two files `task1_training.py` and `task1_testing.py`. In `task1_training.py`, the training of the neural network should be performed, while `task1_testing.py` should test the trained model and print out the achieved accuracy. You can use Tensorflow's `tf.train.Saver` class for this purpose. Please also provide the trained model parameters in the zip file.

## 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 corresponding channel model

$$\underline{y}_i = \mathbf{H}\underline{x}_i + \underline{n}_i, \quad i = 1, \dots, n. \quad (1)$$

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

Further, we associate a length  $mN_t$  bit binary vector with each  $\underline{x} \in \mathcal{X}^{N_t}$  using the mapping  $\chi : \mathcal{X}^{N_t} \rightarrow \{0, 1\}^{mN_t}$ :

$$\chi(\underline{x}) = \underline{b} = (b_1, b_2, \dots, b_{mN_t}). \quad (2)$$

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

$$L_i = \log \left( \frac{P_{B_i|\underline{Y}}(0|\underline{y})}{P_{B_i|\underline{Y}}(1|\underline{y})} \right) = \log \left( \frac{\sum_{\underline{x} \in \mathcal{X}_{i,0}^{N_t}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}{\sum_{\underline{x} \in \mathcal{X}_{i,1}^{N_t}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})} \right), \quad i = 1, \dots, mN_t \quad (3)$$

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

Calculating the log-likelihood values becomes infeasible when the number of transmit antennas  $N_t$  is too large. For example, for 64-QAM and  $N_t = 8$  antennas we have  $|\mathcal{X}^{N_t}| = 64^8 = 2^{48} \approx 10^{14}$ . An approach to circumvent this complexity is to calculate the LLRs as

$$L_i = \log \left( \frac{\sum_{\underline{x} \in \hat{\mathcal{X}}_{i,0}^{N_t}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}{\sum_{\underline{x} \in \hat{\mathcal{X}}_{i,1}^{N_t}} p_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})} \right), \quad i = 1, \dots, mN_t \quad (4)$$

where the respective sets  $\tilde{\mathcal{X}}_{i,b}^{N_t}, b \in \{0, 1\}$  have a much smaller cardinality. However, the question rises how to find these sets. The idea is to use Markov-Chain Monte Carlo (MCMC) sampling to find samples  $\underline{b}$  from  $P_{B|\underline{Y}}(\cdot|y)$  and store them in  $\tilde{\mathcal{X}}_{i,b}^{N_t}, b \in \{0, 1\}$  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  $N_t = N_r = 2$  and  $N_t = N_r = 8$  and 16-QAM. *Hint: You can omit the  $\lambda_i$  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  $\sum_{i=1}^{N_t m} I(B_i; \underline{Y}) = \sum_{i=1}^{N_t m} I(B_i; L_i)$  for the above scenarios for the signal-to-noise ratio (SNR) regime of 0 dB to 15 dB. We define the SNR as

$$\text{SNR} = \frac{\mathbb{E}[\underline{X}^H \underline{X}]}{\sigma^2} = \frac{1}{\sigma^2}.$$

We provide a function `bicm_cap_mc(B, L)` that calculate the BICM capacity via a Monte Carlo approximation. Here, the parameters `B` and `L` are matrices of dimension  $mN_t \times n_s$ , where  $n_s$  is the number of samples. The return variable `R` denotes the achievable rate. You may also 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  $N_t = N_r = 2$ .

### 3 Expectation Maximization

You are given a dataset `EM_data.npy` (load with `np.load`) of noisy receive data points  $y_i \in \mathbb{C}, i = 1, \dots, 10\,000$  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  $\sigma^2$ . In this task, you are supposed to do the following:

- Use the expectation maximization (EM) algorithm to determine the unknown model parameters  $\sigma^2, \Delta$  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.