ITU Artificial Intelligence/Machine Learning in 5G Challenge
ML5G-PHY [beam selection]

Team CERTAIN

Julius Ruseckas, Gediminas Molis, Tomas Krilavičius, Hanna Bogucka

In this report we describe the proposed solution for the beam selection model of the CERTAIN team.

Used resources

For training of the model we employ a computer with a single CUDA capable GPU having 8GB of graphic memory. Training of a model described below takes around 10 min.

Feature extraction

For the beam selection we use features extracted from lidar data and from the GPS coordinates. We do not use image data because experimentation shows that inclusion of image data does not increase the accuracy of the model. Lidar data we take as provided in the baseline features, in the directory `baseline_data/lidar_input`. For the GPS data we record x, y and z coordinates. The feature extraction of the GPS coordinates is performed by a python script `beam_train_frontend.py`.

Data preprocessing

We normalize the GPS coordinates x, y and z. That is we subtract the means and divide by the standard deviation in order to obtain the features with zero mean and unit deviation. The means and standard deviations of the GPS coordinates in the training set we save into the file `coord_train_stats.npz`.

Model

The model consists of two submodules, the first submodule gets the coordinate data as input, the second submodule gets lidar data as input. The output of both submodules is concatenated to form the final feature vector. The final feature vector is then forwarded to the dropout layer with the dropout probability 0.5 and the fully connected layer with softmax activation to form the beam probabilities.

In the submodule responsible for GPS coordinate data a multi layer perceptron (MLP) is applied to the resulting feature vector. For the MLP we use a sequence of fully connected layers with 8, 16, 64 and 256 neurons and ReLU activation functions.
We process the lidar input as 2.5D data; specifically, we use the z coordinate as a channel number. For the second submodule we adopt ResNet-like architecture. The lidar data is first processed by a sequence of 2D convolution layers; each layer is followed by a batch normalization layer and ReLU activation. The first layer has stride 1 in x direction and stride 2 in y direction. We use 3 convolutional layers with 32, 32 and 64 channels.
After this part of the model follows a sequence of residual blocks. We use 2 blocks with 64 channels and 2 blocks with 128 channels. Each residual block contains a sum of shortcut connection and residual branch; the residual branch is added with a weight that is a learnable parameter of the model. The residual branch is formed from 2 convolution layers with batch normalization. The first residual block in each group of 2 blocks has convolution with stride 2. For such a block, in order to make the dimensions of the shortcut connection the same as the dimensions of the residual branch, 2D average pooling with pool size 2 is applied to the shortcut.
Finally, the number of channels in the output of residual blocks is reduced to 8 by a 2D convolution with kernel size 1. The output of the convolution is flattened and used as a feature vector that is then forwarded to the dropout layer with the dropout probability 0.25 and the fully connected layer with 256 neurons and ReLU activation.
The weights of convolutional and linear layers in the model are subjected to L2 regularization with L2 regularization factor 10^{-4}. The model has 1 110 852 trainable and 2 064 non-trainable parameters, making 1 112 916 parameters.

Training

We use the categorical cross entropy loss and trained the model using Adam optimizer with 1cycle learning rate schedule [1]. That is, we increased the learning rate according to a cosine law for the 30% of learning duration up to the maximum learning rate of 10^{-2}. In the last 70% of the learning duration we decreased the learning rate to zero according to a cosine law. The parameter β_1 of the Adam optimizer we similarly decrease to 0.85 and then increase again. The implementation of the scheduler callback is provided in the file `utils.py`. We train the model for 50 epochs and save the weights corresponding to the best top-5 accuracy. In the training we used batch size 32. We obtained 0.68 top-1 accuracy and 0.94 top-5 accuracy on the validation data in Raymobtime s008 dataset. Training of the model is performed by a python script `beam_train_model.py`, weights are saved to the file `my_model_weights.h5`.

[1], Leslie N. Smith, *A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay*, arXiv:1803.09820 [cs.LG] (2018).