Status Report: Bayesian Optimization for Automatically Generating Neural Networks to Model Customer Churn

By Patrick Di Rita

Experimental Setup

In the time between reviewing our proposal's feedback and now, we have been able to come to a much more conclusive decision regarding experimental workflow, evaluation, and implementation. In terms of experimental workflow, we have decided to split the Bayesian optimization step into two independent components: optimization of optimizer hyperparameters and optimization of neural network architecture hyperparameters. Optimizing these two problems separately not only allows for much lower-dimensional search spaces, but also facilitates in-depth analysis and evaluation at each step of the process. Further, we will not be imposing a penalty term for model architecture complexity, as we feel this will greatly diminish exploration at the acquisition step.

With these modifications to our proposed setup in mind, our experimental procedure, as of now, is as follows (we will discuss possible additions to this procedure in a later section of this report):

1. Read in, clean, encode, and split the customer churn data
2. Define control parameters for the optimizer and network architecture
3. Train and evaluate control network using control optimizer to get a baseline score
4. Use Bayesian optimization to obtain optimal optimizer hyperparameters on control network
5. Fix optimizer hyperparameters to those found in (4), and perform Bayesian optimization on architecture hyperparameters

We will then evaluate the trained models from steps (3), (4), and (5) on the test dataset and use their classification accuracy as the performance metric.

Current Progress & Implementation

So far, we have completed steps (1) through (4). We performed 5 rounds of SOBOL random initialization and 15 rounds of Bayesian optimization using the Expected Improvement acquisition function for the Bayesian optimization in (4). We plan to re-visit this step once we have implemented our proposed pooled acquisition function. We are implementing this experiment in Python and have chosen PyTorch for implementing the neural network and optimizer, BoTorch for defining our custom acquisition function, and Ax for performing the Bayesian optimization loop. So far, our implementation has been conducted within a Kaggle notebook due to its ease of use and fast startup, but will be transitioning to a Google Cloud Platform notebook with an 8-core CPU and an Nvidia Quadro P6000 GPU, as the compute power in Kaggle notebooks will be a serious limitation in step (5).

In terms of our results and progress so far, step (1) was quite simple, and our data engineering consisted of simply removing unnecessary features such as customer ID and surname, one-hot encoding categorical features, and standardizing the features by removing the mean and scaling to unit variance.

We then defined our control ANN and Adam optimizer using the control hyperparameters in Table 1 and 2, respectively.

Hidden_Layers	Layer_Sizes	Activation_Functions	Dropout
1	(11, 6, 1)	(ReLU, ReLU, Sigmoid)	10%

Table 1: Control hyperparemeters for ANN architecture. Layer sizes and Activation Functions correspond to (input, hidden, output) layers, respectively. Dropout on input and hidden layers only.

Learning_Rate	Batch_Size	Num_Workers	Weight_Decay	Num_Epochs
.001	10	4	0	100

Table 2: Control hyperparemeters for Adam optimizer.

Learning_Rate	Batch_Size	Num_Workers	Weight_Decay	Num_Epochs
0.0116	100	0	0	100

Table 3: Optimized hyperparemeters for Adam optimizer on the control network

We then performed Bayesian Optimization on the parameters in Table 2 (fixing Num_Epochs at 100) to get the optimal parameterization shown in Table 3. The classification accuracies on the test set for the control network and the control network with optimized hyperparameters were 84% and 85.25%, respectively. Figure 1 shows the model performance over the course of the Bayesian optimization loop, where we achieved our best parameterization at iteration 10.

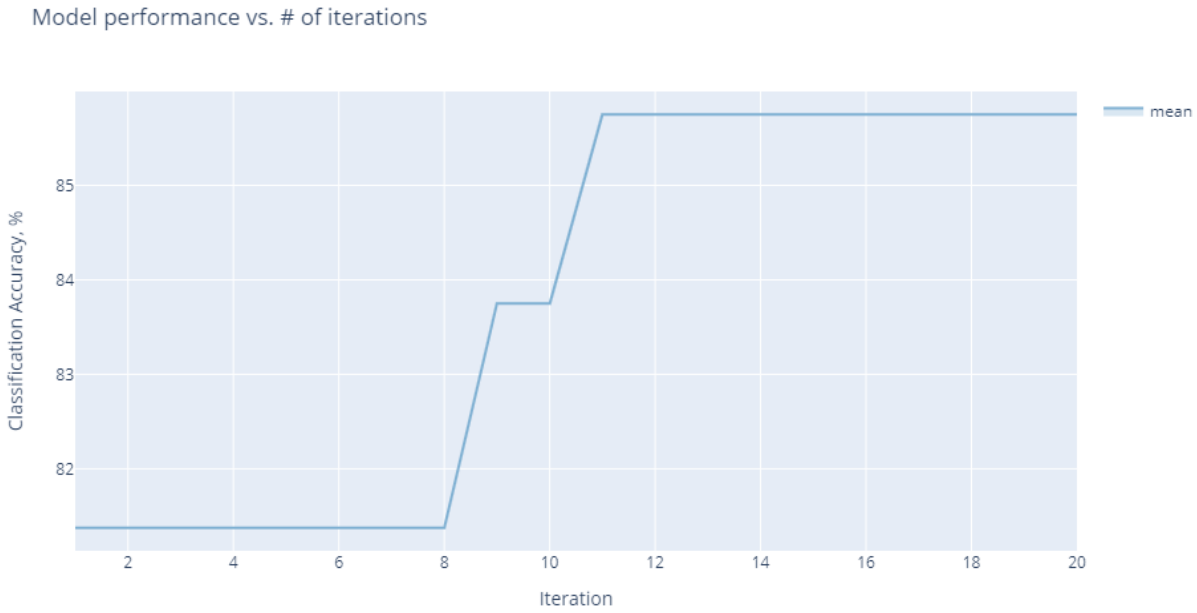

Figure 1: Model performance over the course of Bayesian optimization on Adam hyperparameters

Next Steps & Issues

Our next steps are to implement the pooled acquisition function in BoTorch and do Bayesian optimization loops on the model architecture using both our custom acquisition function and vanilla Expected Improvement. We also plan to re-conduct the optimization of the Adam hyperparameters on our GCP system, as we hypothesize that our model converged to a Num_Workers value of 0 due to Kaggle's limited GPU power.

Next, we plan to do in-depth analysis and visualization of model performance across each step of the experiment, focusing on the effect of acquisition function choice on outcome. We will also visualize the hyperparameter search space by creating contour plots where the x and y axis are hyperparameters and the z axis is classification accuracy.

The main issue we have found with this project is our choice of data. Our simple control model was able to perform fairly well on the data, and it may be nearing the best possible outcome already. This is not a huge problem, as this project is mainly focused on the novel acquisition function scheme and is a proof of concept, but we are interested in continuing our research by extrapolating to more complex data in the future.