Keras Sequential Model Architecture

YU-CHIEH HSU

August, 2025

1 Introduction

This report outlines the architecture and tuning strategies employed in the Churn Prediction Artificial Neural Network. The following section explains the purpose for adopting Keras as the modeling framework, an overview of its structure and highlights the key modification introduced during the development. The next section examines the choice of the initialiser and activation functions for the layers, followed by a discussion of the optimisation process and its parameters. Finally, the report presents an additional section on fine tuning, illustrating how it integrates with the Keras model.

2 Model Structure

The development of this model was informed by *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow* as well as supplementary community discussions and online resources. The Keras deep learning API is widely adopted due to its simplicity and efficiency in building various types of neural networks. Moreover, it is compatible with multiple deep learning frameworks, including TensorFlow, which was subsequently used in this project for evaluation with TensorBoard. Below is the initial model without normalisation and dropouts.

```
def build_model(input_dim:int, units1=32, units2=16, units3=8, lr=1e-3):
    # Initialise ANN
    model = Sequential()

    # Input layer and 1st hidden layer
    model.add(Dense(units1, kernel_initializer='he_uniform', activation='relu',
    input_shape=(input_dim,)))

    # 2nd layer
    model.add(Dense(units2, kernel_initializer='he_uniform', activation='relu')))

    # 3rd layer
    model.add(Dense(units3, kernel_initializer='he_uniform', activation='relu')))

    # Output layer
```

```
model.add(Dense(1, kernel_initializer='glorot_uniform',
activation='sigmoid'))

# Optimise
model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy',
metrics=[tf.keras.metrics.AUC(curve='PR', name='auprc'), 'accuracy'])

return model
```

However, several issues emerged after hyperparameter tuning. The model showed significant overfitting and took nearly 1.75 hours to identify the best parameters. Additionally, the accuracy plateaued at approximately 69%, which was 10% lower than the target level.

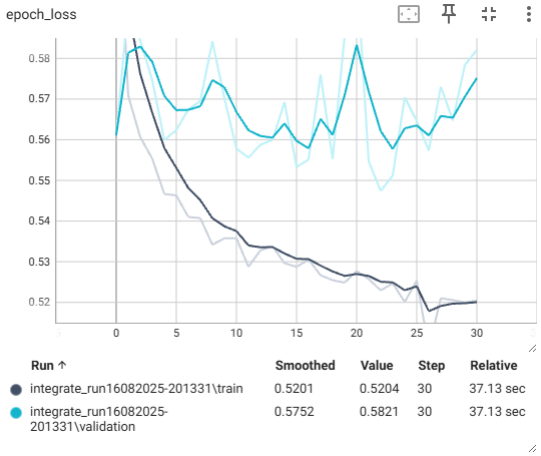

Figure 1: Train validation loss of the first evaluation.

There are several ways to prevent overfitting, the final model was determined by taking these measures:

1. Add Batch Normalisation.
2. Add dropout 0.2.
3. Remove the third layer.
4. Add L2 regularisation.

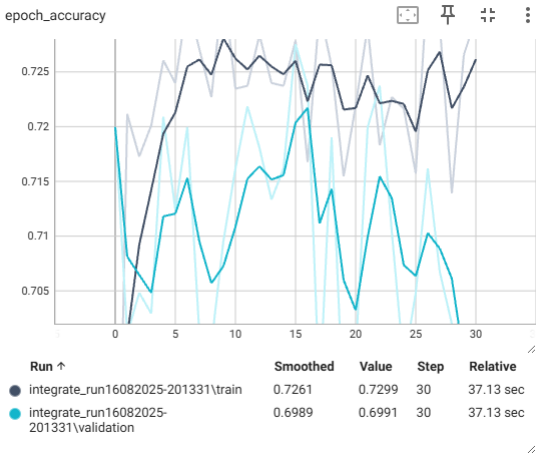

Figure 2: Accuracy fluctuated and converged near 69%, indicating strong overfitting and limited generalisation improvement.

The final model architecture was defined as:

```
def build_model(input_dim:int, units1=32, units2=16, lr=1e-3, l2=1e-4, drop=0.2):
    # Initialise ANN
    model = Sequential()

    # Input layer and 1st hidden layer
    model.add(Dense(units1, kernel_initializer='he_uniform', activation='relu',
    input_shape=(input_dim,), kernel_regularizer=reg.l2(l2)))
    model.add(BatchNormalization())
    model.add(Dropout(drop))

    # 2nd layer
    model.add(Dense(units2, kernel_initializer='he_uniform', activation='relu',
    kernel_regularizer=reg.l2(l2)))
    model.add(BatchNormalization())
    model.add(Dropout(drop))

    # Output layer
    model.add(Dense(1, kernel_initializer='glorot_uniform',
    activation='sigmoid'))

    # Optimise
    model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy',
    metrics=[tf.keras.metrics.AUC(curve='PR', name='auprc'), 'accuracy'])

    return model
```

And the final evaluation is given below:

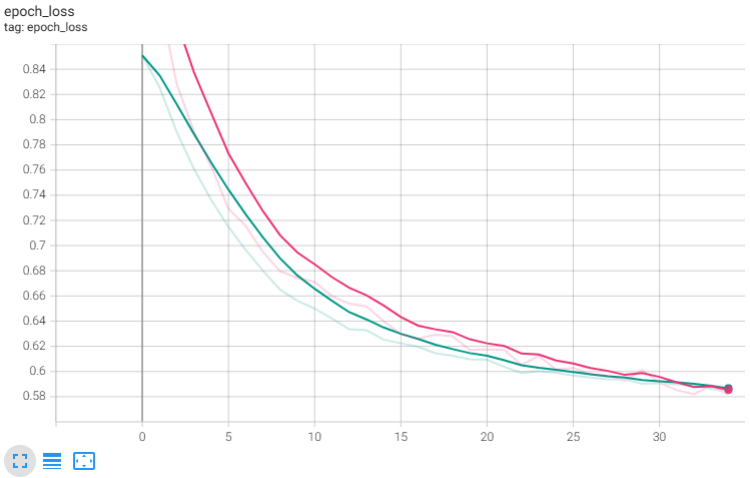

Figure 3: Train validation loss after adding Batch Normalisation, dropout, L2 regularisation, and third hidden layer removed.

The final test accuracy was 0.76 using F1 score as the evaluation threshold with Sciket-Learn Gradient Booster and calibration.

3 Weight Initialiser, Activation, and Compile

1. **Input and Hidden Layers**: He uniform + ReLU
2. **Output Layer**: Glorot uniform + Sigmoid
3. **Optimiser**: Adam
4. **Loss function**: Binary cross-entropy
5. **Metrics**: AUPRC and accuracy

The model was compiled with accuracy as the primary performance metrics, while the area under the precision-recall curve was also monitored to account for class imbalance.

4 Tuning

The number of units were reduced by half to prevent overfitting issues observed during the initial evaluation.

```
def hypermodel(hp, input_dim:int):
    u1 = hp.Int('units1', min_value=16, max_value=64, step=16)
    u2 = hp.Int('units2', min_value=8, max_value=32, step=8)
    lr = hp.Float('lr', 2e-4, 2e-3, sampling='log')
    l2 = hp.Float('l2', 2e-4, 2e-3, sampling='log')
    dr = hp.Float('drop', 0.25, 0.4, step=0.05)
    return build_model(input_dim, u1, u2, lr, l2, dr)
```

The maximum epochs and executions per trial were altered to achieve an average tuning time of approximately 40 minutes, resulting with an AUPRC of 0.6 and a final test accuracy of 0.76 (with F1 optimised threshold) and 0.79 (with accuracy score threshold).

```
def make_tuner(input_dim:int, project_name='krs_hyperband',
directory='hyperband'):
    tuner = kt.Hyperband(
        hypermodel=lambda hp:hypermodel(hp, input_dim),
        max_epochs=30,
        factor=3,
        hyperband_iterations=1,
        objective=Objective('val_auprc', direction='max'),
        executions_per_trial=3,
        directory=directory,
        project_name=project_name,
        overwrite=True)
    return tuner
```

5 Results

- **Best tuning time**: 27 minutes
- **Best AUPRC**: 0.6
- **Best parameters**: "units1": 32, "units2": 8, "lr": 4.42×10^{-4}, "l2": 1.52×10^{-3}, "drop": 0.25
- **Best test accuracy optimised by accuracy (without additional classifiers)**: 0.7932
- **Best test accuracy with accuracy threshold (with additional classifiers)**: 0.7965
- **Best test accuracy with F1 threshold (with additional classifiers)**: 0.7672
- **Best stacked model number**: 1

6 Appendix

6.1 Model Summary

```
[23]: model = cp.model
      model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	448
batch_normalization (BatchNormalization)	(None, 32)	128
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 8)	264
batch_normalization_1 (BatchNormalization)	(None, 8)	32
dropout_1 (Dropout)	(None, 8)	0
dense_2 (Dense)	(None, 1)	9

Total params: 2,485 (9.71 KB)

Trainable params: 801 (3.13 KB)

Non-trainable params: 80 (320.00 B)

Optimizer params: 1,604 (6.27 KB)

Figure 4: Final model summary after fine tuning.

6.2 Healthy Training

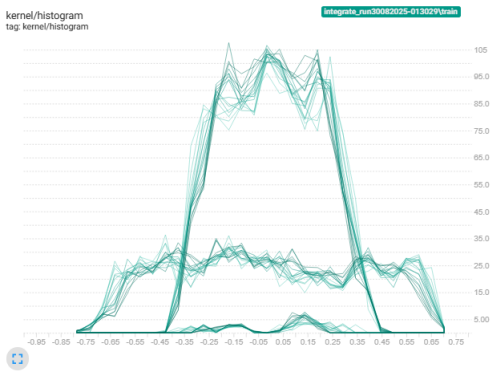

Figure 5: A roughly bell-shaped, symmetric, and stable kernel histogram indicates balanced and healthy learning without weight collapse or explosion.