Case Study VI

Randy Kim, Kati Schuerger, Will Sherman
November 14, 2022

1 Introduction

Objective

The goal of this case study was to build a dense neural network that can predict the existence of a new particle and to maximize the accuracy of the model. The study includes a discussion of some of the decisions made while building the network and how it was determined that the model had finished training. The weight matrix (would be) included as part of the deliverables for this study.

2 Methods

Data

The data were provided by a team of scientists, the client, and is a very large amount of data. The target variable was binary: 1 for detection, 0 for non-detection. A neural network (NN) classification model was selected to accommodate the size of the data. Additionally, as more data is accumulated by the lab, they can be fed into the network in order to continue training the NN layers.

Data pre-processing (EDA)

One-hot-encoding

Initial investigation of the data identified 4 features with only two value options (F9, F13, F17, F21). These features were one-hot-encoded prior to modeling. Once these new features were added, the original feature vectors were dropped from the data.

Scale data

An important step of preparing the data for a neural network model is scaling. Neural networks solve the given task by evaluating and updating weights and slope values for each feature as it passes data through the network. SKLEARN StandardScaler was used to fit and transform the data to prepare it for the model.

Neural network structure

For the model generation and curation, TensorFlow and the Keras API were used. To build the network, the specifications for the structure need to be provided. Next, the model can be compiled. Then the model can be fit and provide predictions.

Structure specifications

The first piece of the network structure is the input layer. This is the number of features that will be passed into the model. After feature engineering there were 32 inputs.

The second piece of the network is hidden layers—in this case, dense layers. These are the worker nodes that will process the data to identify appropriate slopes and weights for each feature. There are two key parts to a dense layer: the number of neurons and the activation function. How layers are connected must also be provided for the network. This study used the sequential method (TensorFlow Keras), each layer goes in order, without skipping any layers.

An activation function must also be specified for each layer of the network. The activation function must be appropriate for the task (classification); if an incorrect activation function is provided, the network will not fail, it will try to solve the model using whatever activation function is given. For this reason, it is important to pick an appropriate function so that the results will be aligned with what we expect. Activation functions applied included: ReLu, Tanh, and sigmoid.

Finally, the last layer of the network is the output layer, which contains the outputs of the model (predictions).

The final specifications of the dense neural network developed for this analysis are as follows:

- Input layer: 32 features
- 3 Hidden layers
- Layer 1: 512 nodes, activation function: ReLu
- Layer 2: 256 nodes, activation function: ReLu
- Layer 3: 256 nodes, activation function: Tanh
- Output layer: 1 target, activation function: sigmoid

Model compilation & fitting

The step for compiling the model included specifying an optimizer, a loss function, and metrics for evaluation of performance on the validate set during training. The optimizer was used by the model to optimize the loss function and to update the weight and bias matrices. This network used the *AdaDelta* optimizer.

In this model, the loss function evaluated over our training epochs to optimize the NN was *binary cross entropy*. Metrics are additional measures of performance to help the network as it is learning on the training and validation data. Metrics included were accuracy, precision, and recall.

The model was fit on training data with a batch size of 10,000 records. The model was permitted to run through as many as 1,000 epochs; however, with early stopping criteria measured on the loss with a minimum delta of 1×10^{-5}, training typically finished between 15 and 30 epochs.

Early stopping criteria allowed the model to monitor the validation loss and mathematically determine when that loss had stopped improving.

Train/validate/test split

Neural networks, like all models, can suffer from overfitting. To combat this, the data were split into three sections: train, validate, test, prior to model fitting. The train set and validate set were

then used to build the model; validation loss was monitored to assess when the model has stopped learning, as determined by early stopping criteria. The test set was then used to evaluate model performance on unseen data. This was done with SKLearn's *train_test_split*.

Model evaluation & Scoring metrics

Several methods were used to evaluate model performance: binary cross-entropy, accuracy, precision, recall, confusion matrix, receiver operating characteristic (ROC). Cross-entropy is a loss function that can be used in TensorFlow to solve classification tasks. Binary cross-entropy is specific to tasks where the target value only has two possible labels, as in this case (0 and 1). This loss is used by the network to optimize model learning. The loss shows how well a model works in terms of prediction error. If the predictions are close to the actuals, the loss will be minimum and if the predictions are away from the actuals, the loss value will maximum. For this case study, we used Binary Cross Entropy which compares each prediction and actual that is either 0 or 1.

Accuracy, precision, and recall were used to evaluate model training. Accuracy, the main metric of interest to the client, confusion matrix, and ROC were used to evaluate performance on unseen data. Accuracy is True Positive + True Negative divided by True Positive + False Positive + True Negative + False Negative. Accuracy can be misleading if used with imbalanced data; target distribution was confirmed to be even split prior to modeling.

We also utilized a confusion matrix and receiver operating characteristic, ROC, curve for tuning and model evaluation. Both evaluation methods will provide the measuring of True Positive Rate and False Positive Rate which will give us a better understanding of how the model is performing.

3 Results

The original distribution of the data for particle detection (*i.e.*, 1) and no detection (*i.e.*, 0) was perfectly symmetric (Figure 1).

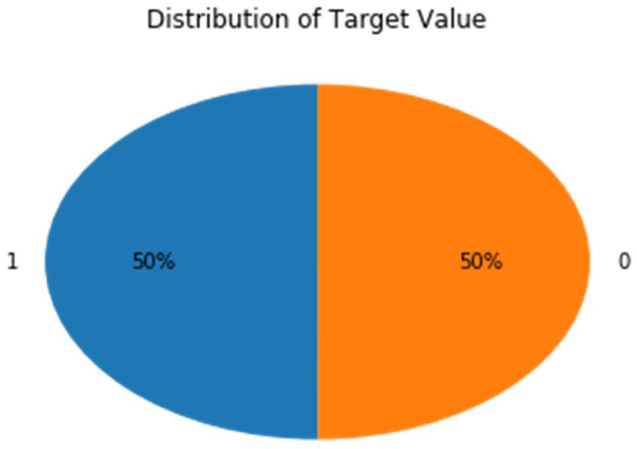

Figure 1. Percent distribution for the number of records where a particle was detected (1) and when a particle was not detected (0).

The loss for both traning and validation data decrease quickly, converging to an average loss of approximately 0.25 and 0.26, respectively (Figure 2a). However, the trend in the validation data indicates that as the number of epoch increases beyond 15 to 20 the variance in loss increases

without appreciable decrease in the loss value. The accuracy between training and validation data exhibits similar behavior and ranges between 86 percent to 88 percent (Figure 2b).

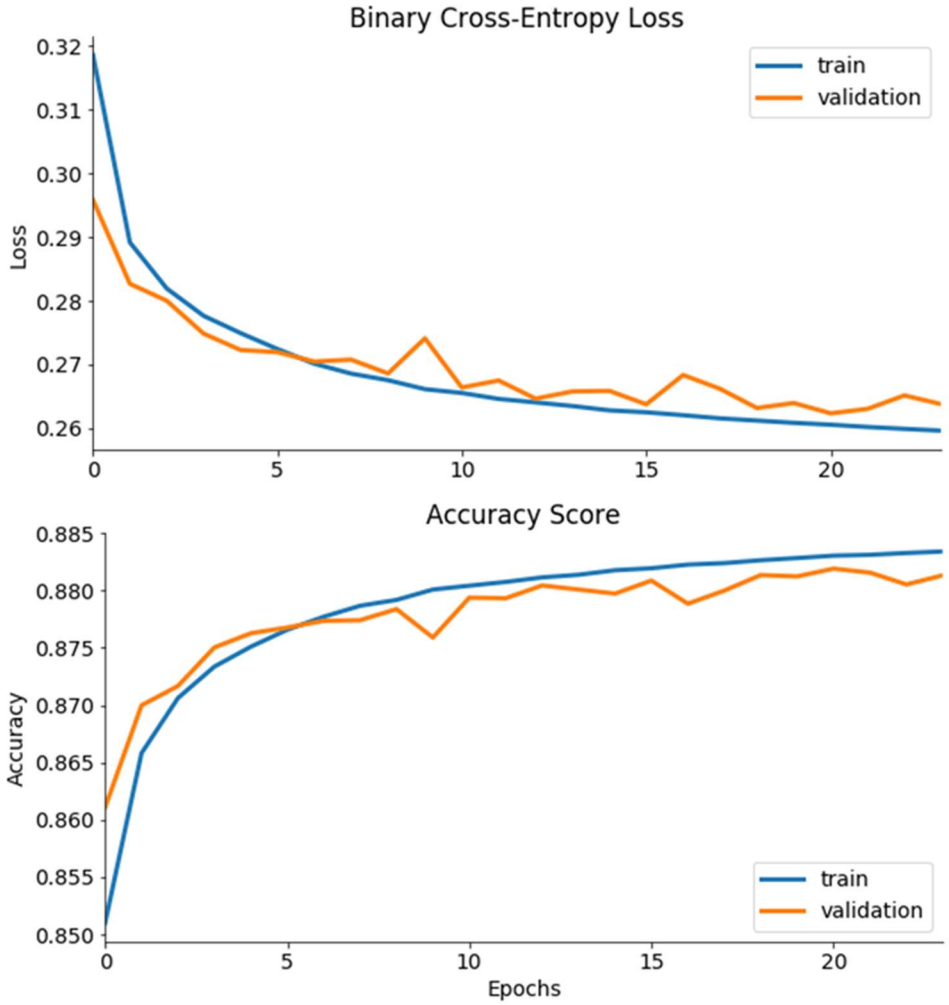

Figure 2. a) Binary cross-entropy loss as a function of the number of epochs; plateau reached after (typically) between 15 and 30 epochs. b) Tracking accuracy score for training and validation datasets as a function of the number of epochs; plateau for validation accuracy indicates no further improvements with increased epochs.

To further evaluate our model, we have a confusion matrix (Figure 3) which details the performance of our model in terms of the Actual vs Predicted values. We normalized each count for predicted and true labels to more easily interpret the matrix. We can see that the *true positive* predictions represented 44.6% of the 1.4 million test instances; the *true negative* predictions represent 43.5%; and the *false positive* and *false negative* instances each account for around 5% of the final data. The rate of model error in the false positive/negative predictions agree with the overall accuracy of the model on unseen data: 88.21%.

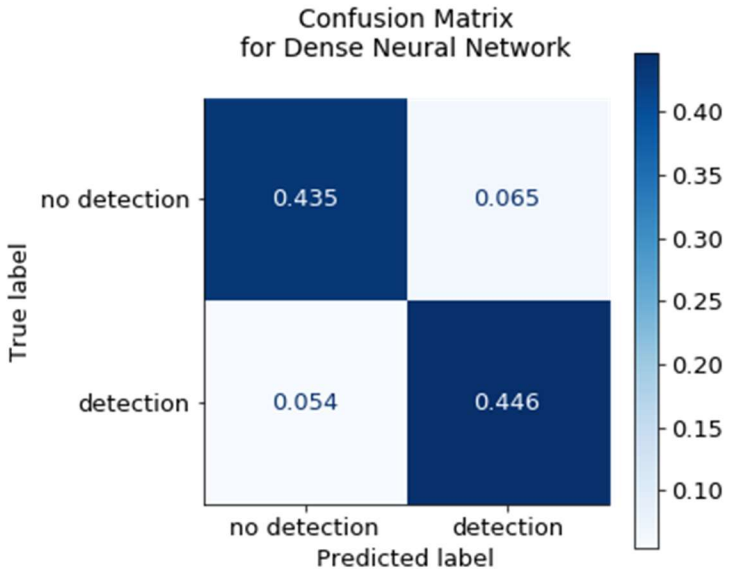

Figure 3. Count-normalized confusion matrix. Values may be considered as proportions of total holdout dataset (1.4 million records).

As a follow-up to the confusion matrix, the receiver operator characteristic (ROC) shows the true positive rate against the false positive rate. When the curve is closer to the top left corner—with an area under the curve (AUC) of 1.000—it indicates that the performance of the model classifies the data better. The ROC AUC for the dense neural network generated as part of this study performed well with an AUC of 0.882 (Figure 4). Incidentally, attempting to gain an increase in the true positive rate may negatively impact accuracy as the incidence of false positives dramatically increases after the elbow.

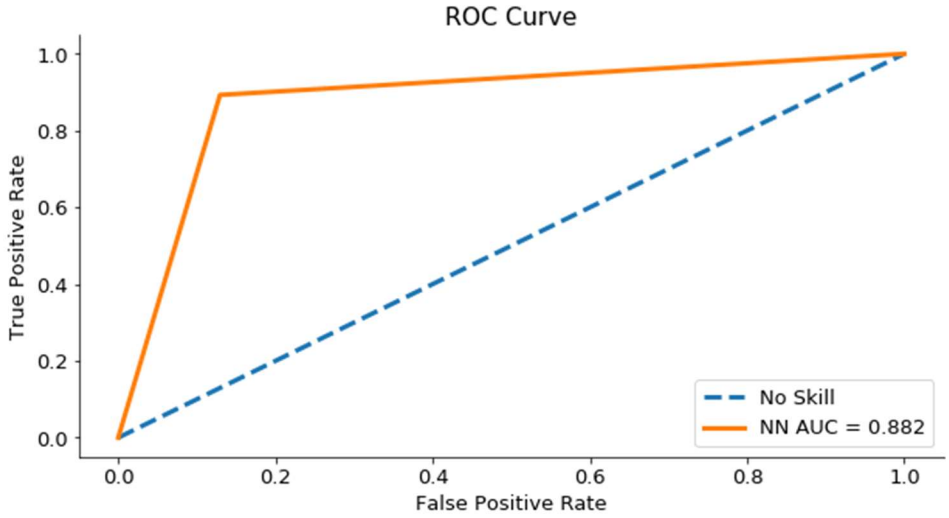

Figure 4. Area under the curve for dense neural network as compared to "No Skill" random guess algorithm.

4 Conclusion

The goal of this case study was to create a dense neural network model that predicts the existence of a new particle with high accuracy. The neural network model herein produced an accuracy of 88.36% on training data, 88.16% on validation data, and 88.21% on test data. Therefore, we can be fairly confident that the model generalizes well to new data.

There was some concern that if the accuracy was inflated over 89 percent, there could be a data leakage; however, because the feature-space is unknown apart from mass, we leave this up to the stakeholders for final determination.

Appendix

Code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam
from matplotlib import rcParams
from tensorflow.keras.callbacks import EarlyStopping

# In[2]:

df = pd.read_csv('all_train.csv.gz', compression='gzip')
df

# In[3]:

df.describe()

# In[4]:

df.info()

# In[5]:

df.isnull().values.any()

# In[6]:

valueCounts = df['# label'].value_counts()
```

```
color = sns.color_palette('tab10')

plt.pie(valueCounts, colors = color, autopct='%.0f%%', labels = ['1', '0'], startangle = 90)
plt.title("Distribution of Target Value")

# In[7]:

df1 = df.drop(['# label'], axis = 1)

df1.hist(bins=100, figsize = (20,15))
plt.show()

# In[8]:

plt.figure(figsize=(20,15))

heatmap = sns.heatmap(df1.corr(), vmin=-1, vmax=1, annot=False, cmap = 'Blues')
heatmap.set_title('Corrleation Heatmap', fontdict={'fontsize':12}, pad=12)

# In[9]:

# https://chrisalbon.com/code/machine_learning/feature_selection/drop_highly_correlated_features/

corr_matrix = df.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]
to_drop

# In[10]:

columns = ['f5', 'f9', 'f13', 'f17', 'f21', 'mass']

for i in columns:
    print("Unique values for Column {} are {}".format(i, df[i].unique()))

# In[11]:

df2 = pd.get_dummies(df, columns=['f9', 'f13', 'f17', 'f21'], prefix=['f9', 'f13', 'f17', 'f21'])
df2

# In[12]:

X = df2.drop(['# label'], axis = 1)
y = df2['# label']

# In[13]:

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# In[14]:

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_state = 12)
x_train, x_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.2, random_state = 12)

# In[15]:

# tf.random.set_seed(12)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(512, input_shape=(32,), activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='tanh'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# In[16]:

model.compile(
    optimizer='adadelta',
    loss='binary_crossentropy',
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
)

# In[17]:

model.summary()

# In[18]:

stops = EarlyStopping(monitor='val_loss', patience=3, min_delta=1e-5)

model_fit = model.fit(x_train, y_train,
                      validation_data=(x_val, y_val),
                      epochs=1000,
                      batch_size=10000,
                      callbacks=[stops])

# In[19]:

rcParams['figure.figsize'] = (10, 5)
rcParams['lines.linewidth'] = 3
rcParams['font.size']=14
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False

# In[20]:

history_df = pd.DataFrame(model_fit.history)
history_df[['loss', 'val_loss']].plot()
plt.title('Binary Cross-Entropy Loss')
plt.ylabel('Loss')
plt.legend(['train','validation'])

history_df = pd.DataFrame(model_fit.history)
history_df[['accuracy', 'val_accuracy']].plot()
plt.title('Accuracy Score')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train','validation'], loc='lower right')

# In[21]:
```

```
pred=model.predict_classes(X_test)

# In[22]:

pred

# In[23]:

from sklearn.metrics import classification_report
print(classification_report(y_test, pred))

# In[24]:

from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred)

# In[25]:

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, pred, normalize='all')
cm = cm.round(decimals=3)

font = {'size'   : 13}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm, display_labels=np.asarray(['no detection','detection']))
fig, ax = plt.subplots(figsize=(5,5))
ax.grid(False)
plt.title('Confusion Matrix\nfor Dense Neural Network\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax, values_format='')

# In[26]:

from sklearn.metrics import roc_auc_score, auc
from sklearn.metrics import roc_curve

ns_probs = [0 for _ in range(len(y_test))]
ns_auc = roc_auc_score(y_test, ns_probs)
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)

roc_log = roc_auc_score(y_test, pred)
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, pred)
area_under_curve = auc(false_positive_rate, true_positive_rate)
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(false_positive_rate, true_positive_rate, label='NN AUC = {:.3f}'.format(area_under_curve))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
# plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))
plt.legend(loc=4)

# In[50]:

with pd.option_context('display.max_rows', 513):
    display(pd.DataFrame(model.get_weights()[0]))

# In[51]:
```

```
with pd.option_context('display.max_rows', 513):
    display(pd.DataFrame(model.get_weights()[2]))

# In[52]:

with pd.option_context('display.max_rows', 513):
    display(pd.DataFrame(model.get_weights()[4]))

# In[53]:

pd.DataFrame(model.get_weights()[5])
```