ANN - Exercise

Construct, train and test an artificial neural network using a dataset of your own choice. Try different settings for two or more hyperparameters and investigate the effect on learning. Hand in a Jupyter notebook which contains your python code and in which you describe your approach and results. Also reflect on the knowledge and skills you acquired on artificial neural networks.

In [1]:

```
# Manually setting the root directory to be Fontys
import os
import sys
# print(os.getcwd())
root_path = os.path.split(os.getcwd())[0]
assert root_path.endswith("/Fontys-ADS"), "The root path does not end with Fonty
s: " + root_path
sys.path.insert(0, root_path)
```

Preparing & Cleaning the data

The dataset I have chosen is the Loan eligibility dataset from kaggle (https://www.kaggle.com/vikasukani/loan-eligible-dataset (https://www.kaggle.com/vikasukani/loan-eligible-dataset)).
I plan to predict whether someone is eligible for a loan.

In [2]:

```
import pandas as pd
import numpy as np

# set the seed
np.random.seed(56)

# the loan_dataset_path.
loan_dataset_path = "dataset/loan-train.csv"

# reads the dataset from csv.
df = pd.read_csv(loan_dataset_path)

# displays the dataset.
df
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

614 rows × 13 columns

By looking at the type of the columns, it can be seen that many columns still need to be cleaned up. </br> The Loan_ID column can be discarded since it only describes the loan as an unique identifier.

In [3]:

```
# drops the Loan_ID column from the dataframe.
df.drop(columns=['Loan_ID'], inplace=True)

# prints the datatypes for each column.
df.dtypes
```

Out[3]:

```
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [4]:

```
def one_hot_encode(df, column_name, drop_first=False):
    # gets the unique values of the column.
    uniques = df[column_name].unique()

    # prints the unique values.
    print(column_name, uniques)

    # checks whether there is a NaN value in the uniques.
    dummy_na = pd.isna(uniques).any()

    # perform one-hot encoding. (drop_first for dummy encoding)
    pa_dummies = pd.get_dummies(df[column_name], prefix=column_name, dummy_na=du
mmy_na, drop_first=drop_first)

    # adds the one-hot encoded columns to the original dataframe.
    df = pd.concat([df, pa_dummies], axis=1)

    # drops the original column.
    return df.drop([column_name], axis=1)


# perform one-hot encoding on categorical columns.
df = one_hot_encode(df, 'Property_Area')
df = one_hot_encode(df, 'Married')
df = one_hot_encode(df, 'Dependents')
df = one_hot_encode(df, 'Education')
df = one_hot_encode(df, 'Gender')
df = one_hot_encode(df, 'Self_Employed')

# performs dummy encoding by dropping the other column.
# this is done to create a single predictable value.
df = one_hot_encode(df, 'Loan_Status', drop_first=True)
```

```
Property_Area ['Urban' 'Rural' 'Semiurban']
Married ['No' 'Yes' nan]
Dependents ['0' '1' '2' '3+' nan]
Education ['Graduate' 'Not Graduate']
Gender ['Male' 'Female' nan]
Self_Employed ['No' 'Yes' nan]
Loan_Status ['Y' 'N']
```

After creating all the categorical columns by one-hot encoding. </br> The last thing that needs to be done is to check whether the other values contain NaN values.

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
ApplicantIncome             0
CoapplicantIncome           0
LoanAmount                 22
Loan_Amount_Term           14
Credit_History             50
Property_Area_Rural         0
Property_Area_Semiurban     0
Property_Area_Urban         0
Married_No                  0
Married_Yes                 0
Married_nan                 0
Dependents_0                0
Dependents_1                0
Dependents_2                0
Dependents_3+               0
Dependents_nan              0
Education_Graduate          0
Education_Not Graduate      0
Gender_Female               0
Gender_Male                 0
Gender_nan                  0
Self_Employed_No            0
Self_Employed_Yes           0
Self_Employed_nan           0
Loan_Status_Y               0
dtype: int64
```

For now, let's just pad the missing data and see what the results are like. </br> If the predictions are really bad, this step could be tried with more attention to the datapoints that have missing data.

In [6]:

```
# interpolates the missing data by padding them with existing values.
df['Loan_Amount_Term'].interpolate('pad', inplace=True)
df['LoanAmount'].interpolate('pad', inplace=True)
df['Credit_History'].interpolate('pad', inplace=True)

# for some reason 1 record does not get padded, this way it will forcefully get
 padded.
df['LoanAmount'].interpolate('bfill', inplace=True)

# let's check if there are any NaN values left.S
print(df.isnull().any())
```

```
ApplicantIncome            False
CoapplicantIncome          False
LoanAmount                 False
Loan_Amount_Term           False
Credit_History             False
Property_Area_Rural        False
Property_Area_Semiurban    False
Property_Area_Urban        False
Married_No                 False
Married_Yes                False
Married_nan                False
Dependents_0               False
Dependents_1               False
Dependents_2               False
Dependents_3+              False
Dependents_nan             False
Education_Graduate         False
Education_Not Graduate     False
Gender_Female              False
Gender_Male                False
Gender_nan                 False
Self_Employed_No           False
Self_Employed_Yes          False
Self_Employed_nan          False
Loan_Status_Y              False
dtype: bool
```

In [7]:

```
# ensures that all values are computable by tensorflow.
df['ApplicantIncome'] = df['ApplicantIncome'].astype(np.float64)
```

In [8]:

```
# initialize gpu
import tensorflow as tf
tf.random.set_seed(56)
# physical_devices = tf.config.experimental.list_physical_devices('GPU')
# tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

In [9]:

```
import os
from datasets.base_dataset import DatasetBase

# the loan dataset class.
class LoanDataset(DatasetBase):
    def __init__(self, df, batch_size, train_percentage, validation_percentage, 
test_percentage):
        # sets the batch size
        self.batch_size = batch_size

        features = tf.cast(df.loc[:, df.columns != 'Loan_Status_Y'].values, tf.f
loat32)
        labels = tf.cast(df.loc[:, 'Loan_Status_Y'].values, tf.bool)

        # sets the data.
        self.data = tf.data.Dataset.from_tensor_slices((features, labels))

        # set the feature length.
        self.feature_length = len(df.columns) - 1

        # shuffles the dataset
        self.shuffle(256)

        # splits the data into train, validation, and test datasets.
        self.split_data_to_train_val_test(self.data, train_percentage, validatio
n_percentage, test_percentage)
```

In [10]:

```
batch_size = 10
train_percentage = 0.6
validation_percentage = 0.2
test_percentage = 0.2
loanDataset = LoanDataset(df, batch_size, train_percentage, validation_percentag
e, test_percentage)
```

```
train: 37 validation: 12 test: 12
```

Creating the ANN model

In [11]:

```
from models.base_model import ModelBase
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import Input, Dropout, Dense

class ANNModel(ModelBase):
    def __init__(self, feature_length, gpu_initialized=False, training=False, li
mit=5000):
        super().__init__(gpu_initialized, training, limit)

        # sets the feature length for input.
        self.feature_length = feature_length

    def predict(self, X):
        # create predictable array, since predicting only works on an array.
        predictable_array = np.expand_dims(X, axis=0)

        # perform prediction and take the first and only prediction out of the p
redictions array.
        prediction = self.model.predict(X, verbose=1)[0]

        return prediction

    def fit(self, training, callbacks, epochs, validation, validation_steps, ste
ps_per_epoch):
        self.model.fit(
            training,
            callbacks=callbacks,
            epochs=epochs,
            validation_data=validation,
            validation_steps=validation_steps,
            steps_per_epoch=steps_per_epoch, verbose=0)

    def compile(self, optimizer='adam', loss='mse', metrics=['mse'], loss_weight
s=[1.0], dense_units=156, drop_out_rate=0.2, show_summary=False):
        inputs = Input((self.feature_length,))

        dense1 = Dense(dense_units, activation='relu', kernel_initializer='gloro
t_uniform')(inputs)
        if self.training:
            dense1 = Dropout(drop_out_rate)(dense1)
        dense2 = Dense(dense_units, activation='relu', kernel_initializer='gloro
t_uniform')(dense1)
        if self.training:
            dense2 = Dropout(drop_out_rate)(dense2)
        dense3 = Dense(dense_units, activation='relu', kernel_initializer='gloro
t_uniform')(dense2)
        if self.training:
            dense3 = Dropout(drop_out_rate)(dense3)
        dense4 = Dense(dense_units, activation='relu', kernel_initializer='gloro
t_uniform')(dense3)
        if self.training:
            dense4 = Dropout(drop_out_rate)(dense4)
        dense5 = Dense(dense_units, activation='relu', kernel_initializer='gloro
t_uniform')(dense4)
        if self.training:
            dense5 = Dropout(drop_out_rate)(dense5)
        outputs = Dense(1, activation='sigmoid', kernel_initializer='glorot_unif
orm')(dense5)
```

```
        # construct the model by stitching the inputs and outputs
        self.model = Model(inputs=inputs, outputs=outputs, name='ANNModel')

        # compile the model
        self.model.compile(optimizer=optimizer, loss=loss, metrics=metrics, loss
_weights=loss_weights)

        if show_summary:
            self.model.summary()
```

In [12]:

```
model = ANNModel(loanDataset.feature_length, training=True, gpu_initialized=True
)
```

In [13]:

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
import datetime

epochs = 512
INIT_LR = 1e-4
opt = Adam(lr = INIT_LR, decay = INIT_LR / epochs)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['mse', 'accura
cy'], show_summary=True)

# current time
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# create the checkpoint path
checkpoint_path = "checkpoints/ANNModel/" + current_time + ".ckpt"

# create logging
log_dir = "logs/ANNModel/" + current_time

# create all callbacks
callbacks = [
  EarlyStopping(patience=50, monitor='val_loss'),
  TensorBoard(log_dir=log_dir, profile_batch=0)
]
```

```
Model: "ANNModel"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 24)]              0         
_________________________________________________________________
dense (Dense)                (None, 156)               3900      
_________________________________________________________________
dropout (Dropout)            (None, 156)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 156)               24492     
_________________________________________________________________
dropout_1 (Dropout)          (None, 156)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 156)               24492     
_________________________________________________________________
dropout_2 (Dropout)          (None, 156)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 156)               24492     
_________________________________________________________________
dropout_3 (Dropout)          (None, 156)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 156)               24492     
_________________________________________________________________
dropout_4 (Dropout)          (None, 156)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 157       
=================================================================
Total params: 102,025
Trainable params: 102,025
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
# fit the model using the training data
results = model.fit(
  training=loanDataset.train_ds,
  callbacks=callbacks,
  epochs=epochs,
  validation=loanDataset.val_ds,
  validation_steps=loanDataset.val_size,
  steps_per_epoch=loanDataset.train_size)

weights_path = 'weights/ANNModel_trained_model_weights'
model.save_weights(weights_path)
```

In [15]:

```
# re initialize the model.
model.training = False
model.compile(optimizer=Adam(lr = 1e-4), loss='binary_crossentropy', metrics=['m
se'], show_summary=False)
model.load_weights(weights_path)

print('\n# Evaluate on test data')
result = model.evaluate(loanDataset.actual_test_ds)
print('test loss, test acc:', result)
res = dict(zip(model.get_metric_names(), result))
print(res)
```

```
WARNING:tensorflow:Inconsistent references when loading the checkpoi
nt into this object graph. Either the Trackable object references in
the Python program have changed in an incompatible way, or the check
point was generated in an incompatible program.

Two checkpoint references resolved to different objects (<tensorflo
w.python.keras.layers.core.Dense object at 0x7f7be810ac10> and <tens
orflow.python.keras.layers.core.Dense object at 0x7f7c2c023a10>).
WARNING:tensorflow:Inconsistent references when loading the checkpoi
nt into this object graph. Either the Trackable object references in
the Python program have changed in an incompatible way, or the check
point was generated in an incompatible program.

Two checkpoint references resolved to different objects (<tensorflo
w.python.keras.layers.core.Dense object at 0x7f7c2c023a10> and <tens
orflow.python.keras.layers.core.Dense object at 0x7f7be8083350>).

# Evaluate on test data
2/2 [==============================] - 0s 773us/step - loss: 0.6460
- mse: 0.2266
test loss, test acc: [0.6460040807723999, 0.22655773162841797]
{'loss': 0.6460040807723999, 'mse': 0.22655773162841797}
```

Hypertuning

Using automatic hypertuning with TensorFlow we can see what the best parameters would be.

In [16]:

```
# Load the Tensorboard notebook extension
%load_ext tensorboard
```

In [17]:

```
# Clear any logs from previous runs
!rm -rf ./logs/
```

In [18]:

```
from tensorboard.plugins.hparams import api as hp
```

In [19]:

```
HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([16, 32, 64, 128, 256]))
HP_DROPOUT = hp.HParam('dropout', hp.Discrete([0.1, 0.2, 0.3, 0.4]))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd', 'rmsprop']))
HP_LOSS = hp.HParam('loss', hp.Discrete(['mse', 'mae', 'binary_crossentropy']))

METRIC_ACCURACY = 'accuracy'

with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
  hp.hparams_config(
    hparams=[HP_NUM_UNITS, HP_DROPOUT, HP_OPTIMIZER, HP_LOSS],
    metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
  )
```

In [20]:

```
def train_test_model(hparams):
    model = ANNModel(loanDataset.feature_length, gpu_initialized=True, training=
True)
    model.compile(
        optimizer = hparams[HP_OPTIMIZER],
        metrics=['accuracy'],
        loss=hparams[HP_LOSS],
        dense_units=hparams[HP_NUM_UNITS],
        drop_out_rate=hparams[HP_DROPOUT])
    results = model.fit(
        training=loanDataset.train_ds,
        callbacks=callbacks + [hp.KerasCallback(log_dir, hparams)],
        epochs=epochs,
        validation=loanDataset.val_ds,
        validation_steps=loanDataset.val_size,
        steps_per_epoch=loanDataset.train_size)

    _, accuracy = model.evaluate(loanDataset.actual_test_ds)
    return accuracy
```

In [21]:

```
hp_results = []
def run(run_dir, hparams):
  with tf.summary.create_file_writer(run_dir).as_default():
    hp.hparams(hparams)  # record the values used in this trial
    accuracy = train_test_model(hparams)
    hp_results.append({'accuracy': accuracy, 'dense_units': hparams[HP_NUM_UNITS
],
      'dropout_rate': hparams[HP_DROPOUT], 'optimizer': hparams[HP_OPTIMIZER],
      'loss': hparams[HP_LOSS] })
    tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

In [22]:

```
session_num = 0

for num_units in HP_NUM_UNITS.domain.values:
  for dropout_rate in HP_DROPOUT.domain.values:
    for optimizer in HP_OPTIMIZER.domain.values:
      for loss in HP_LOSS.domain.values:
        hparams = {
            HP_NUM_UNITS: num_units,
            HP_DROPOUT: dropout_rate,
            HP_OPTIMIZER: optimizer,
            HP_LOSS: loss
        }
        run_name = "run-%d" % session_num
        run('logs/hparam_tuning/' + run_name, hparams)
        session_num += 1
```

```
2/2 [==============================] - 0s 617us/step - loss: 0.6885
- accuracy: 0.6000
2/2 [==============================] - 0s 571us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 558us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 568us/step - loss: 0.6487
- accuracy: 0.6500
2/2 [==============================] - 0s 574us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 702us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 586us/step - loss: 0.5343
- accuracy: 0.8000
2/2 [==============================] - 0s 779us/step - loss: 0.7000
- accuracy: 0.3000
2/2 [==============================] - 0s 628us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 587us/step - loss: 0.6504
- accuracy: 0.6500
2/2 [==============================] - 0s 715us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 551us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 584us/step - loss: 0.7613
- accuracy: 0.5000
2/2 [==============================] - 0s 563us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 564us/step - loss: 0.5500
- accuracy: 0.4500
2/2 [==============================] - 0s 560us/step - loss: 0.5344
- accuracy: 0.8000
2/2 [==============================] - 0s 702us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 545us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 610us/step - loss: 0.6500
- accuracy: 0.6500
2/2 [==============================] - 0s 570us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 732us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 598us/step - loss: 0.6899
- accuracy: 0.6000
2/2 [==============================] - 0s 580us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 530us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 561us/step - loss: 0.6886
- accuracy: 0.6000
2/2 [==============================] - 0s 598us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 557us/step - loss: 0.8000
- accuracy: 0.2000
2/2 [==============================] - 0s 607us/step - loss: 0.4908
- accuracy: 0.8500
2/2 [==============================] - 0s 689us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 529us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 665us/step - loss: 0.6460
```

```
- accuracy: 0.6500
2/2 [==============================] - 0s 552us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 583us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 666us/step - loss: nan - a
ccuracy: 0.4000
2/2 [==============================] - 0s 600us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 649us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 568us/step - loss: 0.5188
- accuracy: 0.8000
2/2 [==============================] - 0s 988us/step - loss: 0.5000
- accuracy: 0.5000
2/2 [==============================] - 0s 544us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 559us/step - loss: 0.4473
- accuracy: 0.8500
2/2 [==============================] - 0s 568us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 559us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 574us/step - loss: 0.5729
- accuracy: 0.7500
2/2 [==============================] - 0s 582us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 620us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 623us/step - loss: 0.8300
- accuracy: 0.4000
2/2 [==============================] - 0s 620us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 566us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 568us/step - loss: 0.5760
- accuracy: 0.7500
2/2 [==============================] - 0s 563us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 560us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 599us/step - loss: 0.4957
- accuracy: 0.8500
2/2 [==============================] - 0s 542us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 554us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 675us/step - loss: 0.5045
- accuracy: 0.8500
2/2 [==============================] - 0s 599us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 555us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 578us/step - loss: 0.6493
- accuracy: 0.6500
2/2 [==============================] - 0s 578us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 551us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 596us/step - loss: nan - a
ccuracy: 0.3000
```

```
2/2 [==============================] - 0s 621us/step - loss: 0.5000
- accuracy: 0.5000
2/2 [==============================] - 0s 572us/step - loss: 0.5000
- accuracy: 0.5000
2/2 [==============================] - 0s 568us/step - loss: 0.6117
- accuracy: 0.7000
2/2 [==============================] - 0s 587us/step - loss: 0.5500
- accuracy: 0.4500
2/2 [==============================] - 0s 623us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 798us/step - loss: 0.5386
- accuracy: 0.8000
2/2 [==============================] - 0s 575us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 604us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 761us/step - loss: 0.5729
- accuracy: 0.7500
2/2 [==============================] - 0s 696us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 583us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 578us/step - loss: 0.6074
- accuracy: 0.7000
2/2 [==============================] - 0s 584us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 533us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 586us/step - loss: 0.6801
- accuracy: 0.6000
2/2 [==============================] - 0s 611us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 559us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 638us/step - loss: 0.5729
- accuracy: 0.7500
2/2 [==============================] - 0s 607us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 550us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 626us/step - loss: 0.5727
- accuracy: 0.7500
2/2 [==============================] - 0s 635us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 674us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 605us/step - loss: 0.7171
- accuracy: 0.5500
2/2 [==============================] - 0s 641us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 617us/step - loss: 0.0500
- accuracy: 0.9500
2/2 [==============================] - 0s 590us/step - loss: 0.4574
- accuracy: 0.9000
2/2 [==============================] - 0s 540us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 562us/step - loss: 0.9000
- accuracy: 0.1000
2/2 [==============================] - 0s 689us/step - loss: 0.6117
- accuracy: 0.7000
2/2 [==============================] - 0s 640us/step - loss: 0.3500
```

```
- accuracy: 0.6500
2/2 [==============================] - 0s 552us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 713us/step - loss: 0.6802
- accuracy: 0.6000
2/2 [==============================] - 0s 535us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 679us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 556us/step - loss: nan - a
ccuracy: 0.2500
2/2 [==============================] - 0s 544us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 615us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 639us/step - loss: 0.5615
- accuracy: 0.7500
2/2 [==============================] - 0s 588us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 650us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 720us/step - loss: 0.5465
- accuracy: 0.8000
2/2 [==============================] - 0s 643us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 557us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 603us/step - loss: nan - a
ccuracy: 0.5000
2/2 [==============================] - 0s 573us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 547us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 575us/step - loss: 0.6383
- accuracy: 0.6500
2/2 [==============================] - 0s 573us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 709us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 552us/step - loss: 0.5729
- accuracy: 0.7500
2/2 [==============================] - 0s 660us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 570us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 556us/step - loss: 0.5729
- accuracy: 0.7500
2/2 [==============================] - 0s 566us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 627us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 665us/step - loss: 0.5144
- accuracy: 0.8000
2/2 [==============================] - 0s 570us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 695us/step - loss: 0.4000
- accuracy: 0.6000
2/2 [==============================] - 0s 570us/step - loss: 0.6749
- accuracy: 0.6000
2/2 [==============================] - 0s 648us/step - loss: 0.2000
- accuracy: 0.8000
```

```
2/2 [==============================] - 0s 582us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 588us/step - loss: 0.4957
- accuracy: 0.8500
2/2 [==============================] - 0s 548us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 606us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 569us/step - loss: 0.7324
- accuracy: 0.5500
2/2 [==============================] - 0s 592us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 545us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 599us/step - loss: 0.6133
- accuracy: 0.7000
2/2 [==============================] - 0s 627us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 588us/step - loss: 0.1000
- accuracy: 0.9000
2/2 [==============================] - 0s 748us/step - loss: nan - a
ccuracy: 0.2500
2/2 [==============================] - 0s 654us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 555us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 683us/step - loss: 0.6529
- accuracy: 0.6500
2/2 [==============================] - 0s 630us/step - loss: 0.5000
- accuracy: 0.5000
2/2 [==============================] - 0s 638us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 649us/step - loss: 0.6151
- accuracy: 0.7500
2/2 [==============================] - 0s 638us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 536us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 579us/step - loss: nan - a
ccuracy: 0.3500
2/2 [==============================] - 0s 578us/step - loss: 0.5500
- accuracy: 0.4500
2/2 [==============================] - 0s 689us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 572us/step - loss: 0.5485
- accuracy: 0.8000
2/2 [==============================] - 0s 646us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 550us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 573us/step - loss: 0.5592
- accuracy: 0.8000
2/2 [==============================] - 0s 554us/step - loss: 0.5500
- accuracy: 0.4500
2/2 [==============================] - 0s 587us/step - loss: 0.8000
- accuracy: 0.2000
2/2 [==============================] - 0s 555us/step - loss: 0.5729
- accuracy: 0.7500
2/2 [==============================] - 0s 577us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 546us/step - loss: 0.4000
```

```
- accuracy: 0.6000
2/2 [==============================] - 0s 590us/step - loss: 0.5230
- accuracy: 0.8000
2/2 [==============================] - 0s 544us/step - loss: 0.6000
- accuracy: 0.4000
2/2 [==============================] - 0s 561us/step - loss: 0.2000
- accuracy: 0.8000
2/2 [==============================] - 0s 628us/step - loss: 0.6477
- accuracy: 0.7000
2/2 [==============================] - 0s 693us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 612us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 580us/step - loss: 0.6115
- accuracy: 0.7000
2/2 [==============================] - 0s 556us/step - loss: 0.8000
- accuracy: 0.2000
2/2 [==============================] - 0s 773us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 631us/step - loss: 0.7312
- accuracy: 0.5500
2/2 [==============================] - 0s 573us/step - loss: 0.3000
- accuracy: 0.7000
2/2 [==============================] - 0s 600us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 597us/step - loss: 0.6739
- accuracy: 0.6000
2/2 [==============================] - 0s 570us/step - loss: 0.1500
- accuracy: 0.8500
2/2 [==============================] - 0s 613us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 622us/step - loss: nan - a
ccuracy: 0.2500
2/2 [==============================] - 0s 567us/step - loss: 0.5500
- accuracy: 0.4500
2/2 [==============================] - 0s 628us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 741us/step - loss: 0.6529
- accuracy: 0.6500
2/2 [==============================] - 0s 581us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 549us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 594us/step - loss: 0.4598
- accuracy: 0.9000
2/2 [==============================] - 0s 547us/step - loss: 0.2500
- accuracy: 0.7500
2/2 [==============================] - 0s 578us/step - loss: 0.3500
- accuracy: 0.6500
2/2 [==============================] - 0s 635us/step - loss: nan - a
ccuracy: 0.3000
2/2 [==============================] - 0s 584us/step - loss: 0.4500
- accuracy: 0.5500
2/2 [==============================] - 0s 551us/step - loss: 0.5500
- accuracy: 0.4500
```

In [23]:

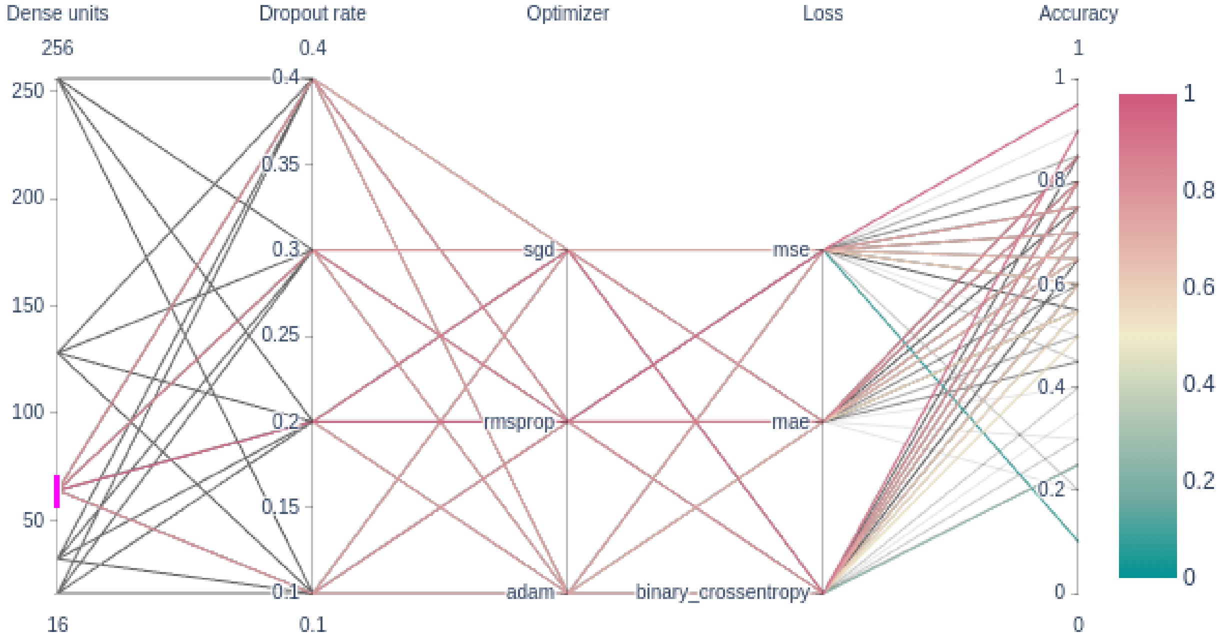
```
# start tensorboard website
#%tensorboard --logdir logs/hparam_tuning
```

In [46]:

```
df = pd.DataFrame(hp_results)

def map_labels(row):
    row['optimizer'] = HP_OPTIMIZER.domain.values.index(row['optimizer'])
    row['loss'] = HP_LOSS.domain.values.index(row['loss'])
    return row

df = df.apply(map_labels, axis='columns')
```

In [53]:

```
import plotly.graph_objects as go

fig = go.Figure(data=
    go.Parcoords(
        line = dict(color = df['accuracy'],
                    colorscale = 'Tealrose',
                    showscale = True,
                    cmin = 0.0,
                    cmax = 1),
        dimensions = list([
            dict(range = [min(HP_NUM_UNITS.domain.values), max(HP_NUM_UNITS.doma
in.values)],
                 label = "Dense units", values = df['dense_units']),
            dict(range = [min(HP_DROPOUT.domain.values), max(HP_DROPOUT.domain.v
alues)],
                 label = 'Dropout rate', values = df['dropout_rate']),
            dict(tickvals = [0, 1, 2],
                 range = [0, len(HP_OPTIMIZER.domain.values)],
                 ticktext = HP_OPTIMIZER.domain.values,
                 label = 'Optimizer', values = df['optimizer']),
            dict(tickvals = [0, 1, 2],
                 range = [0, len(HP_LOSS.domain.values)],
                 ticktext = HP_LOSS.domain.values,
                 label = 'Loss', values = df['loss']),
            dict(range = [0.0, 1],
                 label = 'Accuracy', values = df['accuracy'])])
    )
)
fig.show()
```



In [32]:

```
def reset_labels(row):
    row['optimizer'] = HP_OPTIMIZER.domain.values[int(row['optimizer'])]
    row['loss'] = HP_LOSS.domain.values[int(row['loss'])]
    return row

df = df.apply(lambda row: reset_labels(row), axis='columns')
```

Results

It turned out that the settings I used for the first model in this notebook were not very favourable for the results. A lower amount of dense units will do the job just fine. Also, one thing to note is, that the data set I am using does not have that many records, so the accuracy will not be consistant. Since these models need to learn with a lot more data to be more accurate.

In [33]:

```
df[df.accuracy == df.accuracy.max()]
```

Out[33]:

	accuracy	dense_units	dropout_rate	optimizer	loss
86	0.95	64.0	0.2	rmsprop	mse