Assignment 1- DNN Lab

Step 1: Import Libraries

```
!pip install tensorflow
```

Show hidden output

```
%tensorflow_version X.X
from numpy.random import seed
seed(2)
#from tensorflow import set_random_seed
#set_random_seed(2)
import tensorflow as tf
from tensorflow import keras
from IPython import display
from matplotlib import cm
from matplotlib import gridspec
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.python.data import Dataset
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
print(tf.__version__)
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
2.17.0
```

Step 2: Import Data

```
df = pd.read_csv('hcvdat0.csv')
df.head()
```

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.describe()
df.columns
```

```
Index(['Unnamed: 0', 'Category', 'Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST',
       'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT'],
      dtype='object')
```

```
df.drop(columns=['Unnamed: 0'],axis=1, inplace=True)
df.columns
```

```
Index(['Category', 'Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE',
       'CHOL', 'CREA', 'GGT', 'PROT'],
      dtype='object')
```

```
# prompt: unique values in df['Category']

unique_values = df['Category'].unique()
print(unique_values)
```

```
['0=Blood Donor' '0s=suspect Blood Donor' '1=Hepatitis' '2=Fibrosis'
 '3=Cirrhosis']
```

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

Visualise distributions

```
import matplotlib.pyplot as plt

# extra code – the next 5 lines define the default font sizes
plt.rc('font', size=14)
plt.rc('axes', labelsize=14, titlesize=14)
plt.rc('legend', fontsize=14)
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)

df.hist(bins=50, figsize=(12, 8))

plt.show()
```

Step 3: Data *Preprocessing*

Categorical: Target Encoding: 'Category'

Numerical Data Preprocessing

Variables: 'Category', 'Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT'

Simple imputer(for missing values): 'ALB', 'ALP', 'ALT', 'CHOL', 'PROT' Transformations:

- log transformation: 'ALT', 'AST', 'BIL', 'GGT' (need simple imputer before transformation since it has missing values)
- Quantile transformation: housing_median_age: 'Category', 'Age', 'Sex', 'ALB', 'ALP', 'CHE', 'CHOL', 'CREA', 'PROT' (need simple imputer before transformation since it has missing values)

Pipelines and transformers

```
!pip install sklearn
```

```
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encode
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.1
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->catego
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->
Downloading category_encoders-2.6.4-py2.py3-none-any.whl (82 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82.0/82.0 kB 3.0 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.4
```

```python
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PowerTransformer, OneHotEncoder,  QuantileTransformer, FunctionTransformer, StandardSc
from sklearn.compose import ColumnTransformer
from category_encoders import TargetEncoder
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_absolute_error
```

```python
import pandas as pd
```

Train/Validation Split

```python
# df = pd.read_csv('hcvdat0.csv')
# df.drop(columns=['Unnamed: 0'],axis=1, inplace=True)
df.head()
```

Next steps: Generate code with df View recommended plots New interactive sheet

```
mapping = {
    '0=Blood Donor': 0,
    '0s=suspect Blood Donor': 4,
    '1=Hepatitis': 1,
    '2=Fibrosis': 2,
    '3=Cirrhosis': 3
}
```

```
df['Category'] = df['Category'].map(mapping)
```

```
df['Category'].head()
```

```
# prompt: one-hot ncode category

from tensorflow import keras
from tensorflow.keras import layers

# Assuming 'Category' is your target column
# You can adjust this based on your specific needs
# Create a new column for each category

y = df['Category']

# Convert the target variable to one-hot encoding
y_onehot = keras.utils.to_categorical(y, num_classes=5)
```

```
from sklearn.model_selection import train_test_split

# Features are all columns except 'Target'
X = df.drop('Category', axis=1)
# Target variable is 'Target'
y = df['Category']


# Split the data: 80% for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot, test_size=0.2, random_state=42)

# Print shapes to verify the split
print(f'X_train shape: {X_train.shape}')
print(f'X_test shape: {X_test.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}')

     X_train shape: (492, 12)
```

```
X_test shape: (123, 12)
y_train shape: (492, 5)
y_test shape: (123, 5)
```

```
df["Sex"].value_counts()
```

```
#spac_attribs = ["Category"]
quant_attribs = ['Age', 'ALB', 'ALP', 'CHE', 'CHOL', 'CREA', 'PROT']
log_attribs = ['ALT', 'AST', 'BIL', 'GGT']
cat_attribs = ["Sex"]


#Pipelines
log_pipeline = Pipeline([("impute", KNNImputer(n_neighbors=3)),
                         ("log transform", PowerTransformer(method = 'box-cox'))])
cat_pipeline = Pipeline([("categorical transform", OneHotEncoder(handle_unknown="ignore"))])
quantile_pipeline = Pipeline([("impute", KNNImputer(n_neighbors=3)),
                              ("quantile transformer", QuantileTransformer(n_quantiles=100)),
                              ("standard scaling", StandardScaler())])



preprocessing = ColumnTransformer([
                    #("spatial", "passthrough", spac_attribs),
                    ("quant", quantile_pipeline, quant_attribs),
                    ("log", log_pipeline, log_attribs),
                    ("categorical", cat_pipeline, cat_attribs)
```

```
                ])
```

Step 4: Build Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

https://keras.io/optimizers/

Basic functions

```
#fitting after train-test splitting
def pipelines_fitting(X_train, y_train, preprocessing):
    X_train_processed = preprocessing.fit_transform(X_train, y_train)
    X_df = pd.DataFrame(
        X_train_processed,
        columns=preprocessing.get_feature_names_out(),
        index=X_train.index)
    return X_df
```

```
#Evaluating model pipeline with Linear regression
def evaluate_model(model_pipeline, X, y):
    from sklearn.metrics import mean_absolute_error
    y_pred = model_pipeline.predict(X)
    model_mae = mean_absolute_error(y, y_pred)
    return(model_mae)
```

```
# Preprocess data
X_train_prep = preprocessing.fit_transform(X_train)
X_test_prep = preprocessing.fit_transform(X_test)
X_test_prep = preprocessing.fit_transform(X_test)
```

```
# Create a dataframe from the preprocessed data
X_train_prep = pd.DataFrame(X_train_prep, columns=preprocessing.get_feature_names_out())
X_test_prep = pd.DataFrame(X_test_prep, columns=preprocessing.get_feature_names_out())
```

```
X_train_prep
```

Build Model

```
# Assume X_train_prep is your training data and y_train is one-hot encoded
```

```
num_classes = 5  # Replace with your actual number of classes

baseline_model = keras.Sequential([
    keras.layers.Dense(13, activation='relu', input_shape=(X_train_prep.shape[1],)),
    keras.layers.Dense(num_classes, activation='softmax')  # Softmax in the output layer
])

#optimizer = tf.keras.optimizers.RMSprop(0.001) # Gradient Descent algorithm
#optimizer = tf.keras.optimizers.Adam()

# Compile the model with categorical_crossentropy
baseline_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Display the model's architecture
baseline_model.summary()
```

Fit Model

```
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')
```

```
EPOCHS = 200
b_history = baseline_model.fit(X_train_prep, y_train, epochs=EPOCHS,
                    validation_data= (X_test_prep, y_test), verbose=0,
                    callbacks=[PrintDot()])
```

```
    ....................................................................................................
    ....................................................................................................
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = baseline_model.evaluate(X_test_prep, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
    4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8951 - loss: 0.4141
    Test Accuracy: 0.89
```

```
# Regularized model

baseline_model = keras.Sequential()
baseline_model.add(keras.layers.Dense(64, activation=tf.nn.relu,
                       input_shape=(X_train_prep.shape[1],))),
baseline_model.add(keras.layers.Dense(32, activation=tf.nn.relu,)),
baseline_model.add(keras.layers.Dense(32, activation=tf.nn.relu,)),
baseline_model.add(keras.layers.Dense(32, activation=tf.nn.relu,)),

baseline_model.add(keras.layers.Dense(5, activation='softmax'))

optimizer = tf.keras.optimizers.RMSprop() # Gradient Descent algorithm
optimizer = tf.keras.optimizers.Adam()

baseline_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
baseline_model.summary()
```

Fit Model

```
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

class TrackValidationLoss(keras.callbacks.Callback):
    def __init__(self):
        super(TrackValidationLoss, self).__init__()
        self.lowest_val_loss = np.inf  # Start with infinity

    def on_epoch_end(self, epoch, logs):
        current_val_loss = logs.get('val_loss')
        if current_val_loss < self.lowest_val_loss:
            self.lowest_val_loss = current_val_loss
            print(f'\nLowest validation loss updated: {self.lowest_val_loss:.4f}')

# Create an instance of the custom callback
track_val_loss = TrackValidationLoss()
```

```
EPOCHS = 200
b_history = baseline_model.fit(X_train_prep, y_train, epochs=EPOCHS,
                    validation_data= (X_test_prep, y_test), verbose=0,
                    callbacks=[PrintDot(), track_val_loss])
```

Show hidden output

```
# Evaluate the model on the test set
test_loss, test_accuracy = baseline_model.evaluate(X_test_prep, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8855 - loss: 1.6380
Test Accuracy: 0.87
```

Lowest Validation Error

```
# Print the lowest validation error
lowest_validation_error = min(b_history.history['val_loss'])
print(f'Lowest Validation Error: {lowest_validation_error:.4f}')
```

```
Lowest Validation Error: 0.5241
```

Step 5: Plot Results

```
# Plotting results
train_loss = b_history.history['loss']
val_loss = b_history.history['val_loss']
train_accuracy = b_history.history['accuracy']
val_accuracy = b_history.history['val_accuracy']

# Set the number of epochs for x-axis
epochs_range = range(1, EPOCHS + 1)
```

```
# Create subplots for loss and accuracy
plt.figure(figsize=(12, 5))

# Plot training and validation loss
plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, 'bo-', label='Training Loss', linewidth=1, markersize=1)
plt.plot(epochs_range, val_loss, 'ro-', label='Validation Loss', linewidth=1, markersize=1)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs_range, train_accuracy, 'bo-', label='Training Accuracy', linewidth=1, markersize=1)
plt.plot(epochs_range, val_accuracy, 'ro-', label='Validation Accuracy', linewidth=1, markersize=1)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```

Predictions

```
valpreds = baseline_model.predict_on_batch(X_test_prep)
print(valpreds)
```

Show hidden output

```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(y_test)
```

```
248    0
365    0
432    0
610    3
132    0
602    3
188    0
2      0
271    0
77     0
286    0
244    0
```

```
374    0
137    0
367    0
209    0
101    0
204    0
555    1
441    0
199    0
590    3
299    0
544    1
381    0
522    0
210    0
268    0
611    3
10     0
145    0
304    0
375    0
148    0
181    0
514    0
603    3
55     0
82     0
110    0
290    0
311    0
390    0
131    0
559    1
30     0
570    2
456    0
551    1
403    0
211    0
447    0
533    4
```

```
70     0
260    0
76     0
278    0
```

```
# Plot Weights
nfw = baseline_model.get_weights()[0][0]
y_pos = np.arange(len(nfw))

plt.bar(y_pos, nfw, align='center', alpha=0.5)
```

Regularized Model

```
l1_model = keras.Sequential([
    keras.layers.Dense(15, kernel_regularizer=keras.regularizers.l1(0.1), activation=tf.nn.relu,
                       input_shape=(X_train_prep.shape[1],)),
   # keras.layers.Dense(32, use_bias=True, kernel_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),
    #keras.layers.Dense(32, use_bias=True, kernel_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),
    keras.layers.Dense(10, use_bias=True, kernel_regularizer=keras.regularizers.l1(0.02), activation=tf.nn.relu),
    keras.layers.Dense(5, activation='softmax')

  ])

l1_model.compile(loss='categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['accuracy'])

l2_model = keras.Sequential([
    keras.layers.Dense(15, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu,
                       input_shape=(X_train_prep.shape[1],)),
   # keras.layers.Dense(32, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),
    #keras.layers.Dense(32, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),
    keras.layers.Dense(10, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),
    keras.layers.Dense(5, activation='softmax')
  ])

l2_model.compile(loss='categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
l1_history = l1_model.fit(X_train_prep, y_train, epochs=EPOCHS,
                    validation_data= (X_test_prep, y_test), verbose=0,
                    callbacks=[PrintDot()])
l2_history = l2_model.fit(X_train_prep, y_train, epochs=EPOCHS,
                    validation_data= (X_test_prep, y_test), verbose=0,
                    callbacks=[PrintDot()])
```

```
                callbacks=[PrintDot()])
```

```
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = l1_model.evaluate(X_test_prep, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8220 - loss: 0.6120
Test Accuracy: 0.80
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = l2_model.evaluate(X_test_prep, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8600 - loss: 0.5337
Test Accuracy: 0.84
```

```
def plot_history(histories):
    plt.figure(figsize=(14, 6))

    # Plotting loss
    plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
    for label, history in histories:
        plt.plot(history.history['loss'], label=f'{label} Loss')
        plt.plot(history.history['val_loss'], linestyle='--', label=f'{label} Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid()

    # Plotting accuracy
    plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
    for label, history in histories:
        plt.plot(history.history['accuracy'], label=f'{label} Accuracy')
        plt.plot(history.history['val_accuracy'], linestyle='--', label=f'{label} Val Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid()

    plt.tight_layout()  # Adjust layout to prevent overlap
    plt.show()

# Example usage
plot_history([('baseline', b_history),
               ('L1', l1_history),
               ('L2', l2_history)])
```

Other Models

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU
from keras.utils import to_categorical
```

Batch Normalization

```
# Example model architecture
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape= (X_train_prep.shape[1],)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(5, activation='softmax')  # Softmax for multi-class classification
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Early stopping
early_stopping = keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True)

# Train the model
model.fit(X_train_prep, y_train, epochs=200, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
```

Show hidden output

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test_prep, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8950 - loss: 0.2730
Test Accuracy: 0.88
```

Activation fn- Leaky RElu

```
# Build model
model = Sequential()
model.add(Dense(128, input_shape=(X_train_prep.shape[1],)))
model.add(LeakyReLU(alpha=0.01))
model.add(Dense(64))
model.add(LeakyReLU(alpha=0.01))
model.add(Dense(5, activation='softmax'))  # 5 classes

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
model.fit(X_train_prep, y_train, epochs=200, batch_size=32, validation_split=0.2)
```

Show hidden output

```
# Evaluate model
loss, accuracy = model.evaluate(X_test_prep, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8952 - loss: 0.9446
Test Loss: 0.9902909398078918, Test Accuracy: 0.8943089246749878
```

Increased depth, Leaky RElu, Batch Normalisation

```
# Build model
model = Sequential()
model.add(Dense(128, input_shape=(X_train_prep.shape[1],)))
model.add(LeakyReLU(alpha=0.01))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(64))
model.add(LeakyReLU(alpha=0.01))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(32))
model.add(LeakyReLU(alpha=0.01))
model.add(BatchNormalization())

model.add(Dense(5, activation='softmax'))  # 5 classes

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
model.fit(X_train_prep, y_train, epochs=200, batch_size=32, validation_split=0.2)
```

Show hidden output

Show hidden output

```
# Evaluate model
loss, accuracy = model.evaluate(X_test_prep, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9067 - loss: 0.3837
Test Loss: 0.42674562335014343, Test Accuracy: 0.8943089246749878
```

Conclusion

- The accuracy the model using a naïve approach: 0.89
- The accuracy of the best model: 0.894

**Comparison:* *

- Baseline model: 0.89
- Leaky RElu: 0.894
- Batch Normalization: 0.88
- Increase depth, Leaky RElu, Batch Normalization: 0.894

Batch normalization gives an accuracy less than the baseline model, however other models give a better accuracy than the baseline model. These models may not be perfect, since the comparison of test and validation accuracies, overfitting are supposed to be addressed