```
#%% md
# Taylor Imhof
# Bellevue University | DSC 650
# Assignment05
# Date: 6/26/2022
#%% md
# Movie Review Classifier
#%%
# import imdb dataset from keras
from tensorflow.keras.datasets import imdb
#%%
# load imdb movie dataset from TF
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
#%%
train_data[:5]
#%%
train_labels[:5]
#%%
# word index is restricted to < 10,000
# set by passing num_words param to imdb.load_data()
max([max(sequence) for sequence in train_data])
#%%
# decoding reviews back to text
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value,key) for (key,value) in word_index.items()]
)
decoded_review = " ".join(
    [reverse_word_index.get(i-3,"?") for i in train_data[0]]
)
#%%
type(decoded_review)
decoded_review
#%% md
## Preparing the data
 - pad lists to have same length
 - change to integer tensor of shape(samples, max_length)
 - embedding to start capable model
 - multi-hot encode to turn into vectors of 0s and 1s
#%%
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i,j] = 1
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
#%%
# vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
#%% md
## Building the Model
#%%
# model definition
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
#%%
# compiling the model
model.compile(
    optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
#%%
# set aside validation data to test how model performs on "new" data
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
#%%
# training the model
history = model.fit(
    partial_x_train,
    partial_y_train,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
#%%
# plotting training and validation loss
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training Loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation Loss')
plt.title('Training & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
#%%
# plotting training and validation accuracy
plt.clf() # clears figure
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
#%%
# retrain the model without four epochs as to avoid overfitting/overoptimizing
model = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
model.compile(
    optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.fit(x_train, y_train, epochs=4, batch_size=128)
#%%
results = model.evaluate(x_test, y_test)
#%%
print(f'Test Loss:\t\t{results[0]:.4f}\nTest Accuracy:\t{results[1]:.4f}')
#%%
# predict on "new" data
model.predict(x_test)
#%% md
# News Classifier
#%%
# load reuters news dataset from keras
from tensorflow.keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)
#%%
# check length of training and testing data
print(len(train_data))
print(len(test_data))
#%%
# decode newswire back to text
word_index = reuters.get_word_index()
reverse_word_index = dict(
    [(value,key) for (key,value) in word_index.items()]
)
decoded_newswire = " ".join(
    [reverse_word_index.get(i - 3, '?') for i in train_data[0]]
)
decoded_newswire
#%% md
## Preparing the Data
#%%
# vectorize input data
x_train = vectorize_sequences(train_data)
```

```
x_test = vectorize_sequences(test_data)
#%%
# vectors labels using one-hot encoding
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
y_train = to_one_hot(train_labels)
y_test = to_one_hot(test_labels)
#%%
# one-hot encoding implementation built into keras
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)
#%%
# have to use larger units than movie review as there are much
# more dimensions for the different cats of news articles
model = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(46, activation='softmax')
])
#%%
# compiling the model
model.compile(
    optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
#%%
# set aside validation set
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = y_train[:1000]
partial_y_train = y_train[1000:]
#%%
# training the model
history = model.fit(
    partial_x_train,
    partial_y_train,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
#%% md
## Plotting Losses To Determine Best # of Epochs
#%%
# plot train and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.lenend()
plt.show()
#%%
# plotting train/validation accuracy
plt.clf()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.lenged()
plt.show()
#%% md
It would appear that the values seem to level/taper around 9 epochs
#%%
# retrain model with "better" # of epochs
model = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(46, activation='softmax')
])
model.compile(
    optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(
    x_train,
    y_train,
    epochs=9,
    batch_size=512
)
results = model.evaluate(x_test, y_test)
#%%
print(f'Test loss:\t{results[0]:.4f}\nTest Acc:\t{results[1]:.4f}')
#%%
# check accuracy of random baseline
import copy
test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
hits_array = np.array(test_labels) == np.array(test_labels_copy)
hits_array.mean()
#%% md
## Generate predictions on new data
#%%
preds = model.predict(x_test)
#%%
preds[0].shape
#%%
# coefficients of vector values should be 1 as it forms a probability distribution
np.sum(preds[0])
#%%
# class with highest probability
np.argmax(preds[0])
#%% md
## Housing Price Regression Model
#%%
# load boston housing price data from keras
from tensorflow.keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
#%%
print(train_data.shape)
print(test_data.shape)
#%% md
## Preparing the data
#%%
# feature-wise normalization to account for different ranges of measured observations
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
#%% md
## Model Definition
#%%
#
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1) # no activation as it is a linear layer
    ])
    model.compile(
        optimizer='rmsprop',
        loss='mse',
        metrics=['mae']
    )
    return model
#%% md
## Validation via K-fold Validation
#%%
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print(f'Processing fold #{i}')
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
```

```python
        axis=0
    )
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0
    )
    model = build_model()
    model.fit(
        partial_train_data,
        partial_train_targets,
        epochs=num_epochs,
        batch_size=16,
        verbose=0
    )
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
#%%
all_scores
#%%
np.mean(all_scores)
#%%
# same implementation but saving validation logs
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print(f'Processing fold #{i}')
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0
    )
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0
    )
    model = build_model()
    history = model.fit(
        partial_train_data,
        partial_train_targets,
        validation_data=(val_data, val_targets),
        epochs=num_epochs,
        batch_size=16,
        verbose=0
    )
    mae_history = history.history['val_mae']
    all_mae_histories.append(mae_history)
#%%
# build history of successive mean k-fold validation scores
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)
]
#%%
# plot validation scores
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
#%%
# omit first few data points and re-draw the plot
truncated_mae_history = average_mae_history[10:]
plt.plot(range(1, len(truncated_mae_history) + 1), truncated_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
#%%
## Training Final Model
#%%
# retrain model with "better" # of epochs
model = build_model()
model.fit(
    train_data,
    train_targets,
    epochs=130,
    batch_size=16,
    verbose=0
)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
#%%
test_mae_score
#%% md
## Generate Predictions on New Data
#%%
preds = model.predict(test_data)
preds[0]
```