Building deep learning models with keras

Puteaux, Fall/Winter 2020-2021

```
################################
##                            ##
##  Deep Learning in Python   ##
##                            ##
################################
```

```
§1 Introduction to Deep Learning in Python
```

```
§1.3 Building deep learning models with keras
```

1 Creating a keras model

1.1 What are the model-building steps?

- The Keras workflow has four steps:
  - specify Architecture
  - compile
  - fit
  - predict

1.2 Code of model specification:

[1]:
```python
import numpy as np
from keras.layers import Dense
from keras.models import Sequential

predictors = np.loadtxt('ref1. Hourly wages predictors data.csv',
                        delimiter=',')
n_cols = predictors.shape[1]

model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(n_cols, )))
model.add(Dense(100, activation='relu'))
model.add(Dense(1))
```

1.3 Practice question for understanding the data:

- It will be started soon to building models in Keras to predict wages based on various professional and demographic factors by the next steps. Before starting building a model, it's good to understand the data by performing some exploratory analysis.
- It is recommended to use the `.head()` and `.describe()` methods in the IPython Shell to quickly overview the DataFrame.
- The target variable which will be predicting is `wage_per_hour`. Some of the predictor variables are binary indicators, where a value of `1` represents `True`, and `0` represents `False`.
- Of the nine predictor variables in the DataFrame, how many are binary indicators? The min and max values, as shown by `.describe()` will be informative here. How many binary indicator predictors are there?

  ☐ 0.

  ☐ 5.

  ☒ 6.

▶ **Package pre-loading:**

[2]:
```python
import pandas as pd
```

▶ **Data pre-loading:**

[3]:
```python
df = pd.read_csv('ref2. Hourly wages.csv')
```

▶ **Question-solving method:**

[4]:
```python
df.head()
```

[4]:
```
   wage_per_hour  union  education_yrs  experience_yrs  age  female  marr  \
0           5.10      0              8              21   35       1     1
1           4.95      0              9              42   57       1     1
2           6.67      0             12               1   19       0     0
3           4.00      0             12               4   22       0     0
4           7.50      0             12              17   35       0     1

   south  manufacturing  construction
0      0              1             0
1      0              1             0
2      0              1             0
3      0              0             0
4      0              0             0
```

[5]:
```python
df.describe()
```

[5]:
```
       wage_per_hour       union  education_yrs  experience_yrs         age  \
count     534.000000  534.000000     534.000000      534.000000  534.000000
```

```
mean        9.024064    0.179775     13.018727        17.822097   36.833333
std         5.139097    0.384360      2.615373        12.379710   11.726573
min         1.000000    0.000000      2.000000         0.000000   18.000000
25%         5.250000    0.000000     12.000000         8.000000   28.000000
50%         7.780000    0.000000     12.000000        15.000000   35.000000
75%        11.250000    0.000000     15.000000        26.000000   44.000000
max        44.500000    1.000000     18.000000        55.000000   64.000000

           female        marr       south  manufacturing  construction
count  534.000000  534.000000  534.000000     534.000000    534.000000
mean     0.458801    0.655431    0.292135       0.185393      0.044944
std      0.498767    0.475673    0.455170       0.388981      0.207375
min      0.000000    0.000000    0.000000       0.000000      0.000000
25%      0.000000    0.000000    0.000000       0.000000      0.000000
50%      0.000000    1.000000    0.000000       0.000000      0.000000
75%      1.000000    1.000000    1.000000       0.000000      0.000000
max      1.000000    1.000000    1.000000       1.000000      1.000000
```

[6]:

```
cols = df.columns
count = 0
for i in range(len(cols)):
    if ((df.iloc[:, i].unique()[0] in [0, 1])
            and (df.iloc[:, i].unique()[1] in [0, 1])):
        count += 1
    else:
        pass
print('There are {} binary indicator predictors here.'.format(count))
```

```
There are 6 binary indicator predictors here.
```

1.4 Practice exercises for creating a Keras model:

▶ **Package pre-loading:**

[7]:

```
import pandas as pd
```

▶ **Data pre-loading:**

[8]:

```
df = pd.read_csv('ref2. Hourly wages.csv')

target = df.iloc[:, 0].to_numpy()
predictors = df.iloc[:, 1:].to_numpy()
```

▶ **Model specifying practice:**

[9]:

```
# Import necessary modules
import keras
from keras.layers import Dense
```

```
from keras.models import Sequential

# Save the number of columns in predictors: n_cols
n_cols = predictors.shape[1]

# Set up the model: model
model = Sequential()

# Add the first layer
model.add(Dense(50, activation='relu', input_shape=(n_cols, )))

# Add the second layer
model.add(Dense(32, activation='relu'))

# Add the output layer
model.add(Dense(1))
```

2 Compiling and fitting a model

2.1 Why is it necessary to compile the model?

- Specify the optimizer:
  - many options and mathematically complex
  - `adam` is usually a good choice
- Loss function:
  - `mean_squared_error` is common for regression

2.2 Code of compiling a model:

[10]:
```
import numpy as np
from keras.layers import Dense
from keras.models import Sequential

predictors = np.loadtxt('ref1. Hourly wages predictors data.csv',
                        delimiter=',')
n_cols = predictors.shape[1]

model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(n_cols, )))
model.add(Dense(100, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

2.3 What is fitting a model?

- Apply backpropagation and gradient descent with the data to update the weights.
- Scale data before fitting can ease optimization.

2.4 Code of fitting a model:

[11]:
```
target = np.loadtxt('ref3. Hourly wages target data.csv', delimiter=',')

model.fit(predictors, target)
```

```
17/17 [==============================] - 0s 21ms/step - loss: 92.8207
```

[11]:
```
<tensorflow.python.keras.callbacks.History at 0x7fd96d97f450>
```

2.5 Practice exercises for compiling and fitting a model:

▶ **Package pre-loading:**

[12]:
```
import pandas as pd
```

▶ **Data pre-loading:**

[13]:
```
df = pd.read_csv('ref2. Hourly wages.csv')

predictors = df.iloc[:, 1:].to_numpy()
```

▶ **Model compiling practice:**

[14]:
```
# Import necessary modules
import keras
from keras.layers import Dense
from keras.models import Sequential

# Specify the model
n_cols = predictors.shape[1]
model = Sequential()
model.add(Dense(50, activation='relu', input_shape=(n_cols, )))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Verify that model contains information from compiling
print("Loss function: " + model.loss)
```

```
Loss function: mean_squared_error
```

▶ **Data re-pre-loading:**

[15]:
```
target = df.iloc[:, 0].to_numpy()
```

▶ **Model fitting practice:**

[16]:
```
# Import necessary modules
import keras
from keras.layers import Dense
from keras.models import Sequential

# Specify the model
n_cols = predictors.shape[1]
model = Sequential()
model.add(Dense(50, activation='relu', input_shape=(n_cols, )))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Fit the model
model.fit(predictors, target)
```

```
17/17 [==============================] - 0s 18ms/step - loss: 23.7921
```

[16]:
```
<tensorflow.python.keras.callbacks.History at 0x7fd96e48f550>
```

3 Classification models

3.1 How to compile the classification model with Keras?

- Use `'categorical_crossentropy'` loss function, which is similar to log loss, but lower is better.
- Add `metrics = ['accuracy']` to compile step for easy-to-understand diagnostics.
- The output layer has a separate node for each possible outcome and uses `'softmax'` activation.

3.2 How to transform the target value into categorical?

shot_clock	dribbles	touch_time	shot_dis	close_def_dis	shot_result
10.8	2	1.9	7.7	1.3	1
3.4	0	0.8	28.2	6.1	0
0	3	2.7	10.1	0.9	0
10.3	2	1.9	17.2	3.4	0

shot_result
1
0
0
0

→

Outcome 0	Outcome 1
0	1
1	0
1	0
1	0

3.3 Code of classification:

[17]:
```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential


def data_preparation(df):
    df = df.reindex(columns=[
        'SHOT_CLOCK', 'DRIBBLES', 'TOUCH_TIME', 'SHOT_DIST', 'CLOSE_DEF_DIST',
        'SHOT_RESULT'
    ])
    df['SHOT_CLOCK'] = df['SHOT_CLOCK'].fillna(0)
    df['SHOT_RESULT'].replace('missed', 0, inplace=True)
    df['SHOT_RESULT'].replace('made', 1, inplace=True)
    df.columns = df.columns.str.lower()
    return df
```

[18]:
```
from keras.utils.np_utils import to_categorical

data = pd.read_csv('ref5. Basketball shot log.csv')

data = data_preparation(data)

predictors = data.drop(['shot_result'], axis=1).to_numpy()
n_cols = predictors.shape[1]
target = to_categorical(data.shot_result)

model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(n_cols, )))
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(predictors, target)
```

```
4003/4003 [==============================] - 5s 1ms/step - loss: 0.6655 -
accuracy: 0.6068
```

[18]: <tensorflow.python.keras.callbacks.History at 0x7fd954a66590>

3.4 Practice question for understanding the classification data:

- To start modeling with a new dataset for a classification problem. This data includes information about passengers on the Titanic. The predictors such as `age`, `fare`, and where each passenger embarked to could be used to predict who will survive. This data is from a tutorial on data science competitions. There are descriptions of the features.
- It's smart to review the maximum and minimum values of each variable to ensure the data isn't misformatted or corrupted. What was the maximum age of passengers on the Titanic? Use the `.describe()` method in the IPython Shell to answer this question.

☐ 29.699.

☒ 80.

☐ 891.

☐ It is not listed.

▶ **Package pre-loading:**

[19]:
```
import pandas as pd
```

▶ **Data pre-loading:**

[20]:
```
df = pd.read_csv('ref6. Titanic.csv')
```

▶ **Question-solving method:**

[21]:
```
df.head()
```

[21]:
```
   survived  pclass   age  sibsp  parch     fare  male  age_was_missing  \
0         0       3  22.0      1      0   7.2500     1            False
1         1       1  38.0      1      0  71.2833     0            False
2         1       3  26.0      0      0   7.9250     0            False
3         1       1  35.0      1      0  53.1000     0            False
4         0       3  35.0      0      0   8.0500     1            False

   embarked_from_cherbourg  embarked_from_queenstown  \
0                        0                         0
1                        1                         0
2                        0                         0
3                        0                         0
```

```
4                       0                         0

   embarked_from_southampton
0                          1
1                          0
2                          1
3                          1
4                          1
```

[22]:
```
df['age'].describe()
```

[22]:
```
count    891.000000
mean      29.699118
std       13.002015
min        0.420000
25%       22.000000
50%       29.699118
75%       35.000000
max       80.000000
Name: age, dtype: float64
```

[23]:
```
max_age = int(df['age'].max())
print('The maximum age of passengers on the Titanic is {}.'.format(max_age))
```

```
The maximum age of passengers on the Titanic is 80.
```

3.5 Practice exercises for classification models:

▶ **Package pre-loading:**

[24]:
```
import pandas as pd
```

▶ **Data pre-loading:**

[25]:
```
df = pd.read_csv('ref6. Titanic.csv')

df['age_was_missing'].replace(False, 0, inplace=True)
df['age_was_missing'].replace(True, 1, inplace=True)

predictors = df.drop(['survived'], axis=1).to_numpy()
n_cols = predictors.shape[1]
```

▶ **Classification models practice:**

[26]:
```
# Import necessary modules
import keras
from keras.layers import Dense
from keras.models import Sequential
```

```
from keras.utils import to_categorical

# Convert the target to categorical: target
target = to_categorical(df.survived)

# Set up the model
model = Sequential()

# Add the first layer
model.add(Dense(32, activation='relu', input_shape=(n_cols, )))

# Add the output layer
model.add(Dense(2, activation='softmax'))

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

# Fit the model
model.fit(predictors, target)
```

```
28/28 [==============================] - 0s 9ms/step - loss: 4.1567 - accuracy:
0.5994
```

[26]:
```
<tensorflow.python.keras.callbacks.History at 0x7fd9533e1950>
```

4 Using models

4.1 How to use models?

- Save.
- Reload.
- Make predictions.

4.2 Code of saving, reloading, and using the model reloaded:

[27]:
```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from keras.utils.np_utils import to_categorical

data = pd.read_csv('ref5. Basketball shot log.csv')


def data_preparation(df):
```

```
    df = df.reindex(columns=[
        'SHOT_CLOCK', 'DRIBBLES', 'TOUCH_TIME', 'SHOT_DIST', 'CLOSE_DEF_DIST',
        'SHOT_RESULT'
    ])
    df['SHOT_CLOCK'] = df['SHOT_CLOCK'].fillna(0)
    df['SHOT_RESULT'].replace('missed', 0, inplace=True)
    df['SHOT_RESULT'].replace('made', 1, inplace=True)
    df.columns = df.columns.str.lower()
    return df


df = data_preparation(data)
predictors = df.drop(['shot_result'], axis=1).to_numpy()
n_cols = predictors.shape[1]
target = to_categorical(df.shot_result)

model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(n_cols, )))
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(predictors, target)
```

```
4003/4003 [==============================] - 5s 1ms/step - loss: 0.6636 -
accuracy: 0.6062
```

[27]:
```
<tensorflow.python.keras.callbacks.History at 0x7fd953553450>
```

[28]:
```
from keras.models import load_model

model.save('ref7. Model file.h5')
my_model = load_model('ref7. Model file.h5')

predictions = my_model.predict(predictors)
probability_true = predictions[:, 1]
probability_true
```

[28]:
```
array([0.40594074, 0.32399714, 0.29844287, …, 0.39887083, 0.32704052,
       0.3955048 ], dtype=float32)
```

[29]:
```
my_model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 100)               600

_________________________________________________________________
dense_22 (Dense)             (None, 100)               10100

_________________________________________________________________
dense_23 (Dense)             (None, 100)               10100

_________________________________________________________________
dense_24 (Dense)             (None, 2)                 202
=================================================================
Total params: 21,002
Trainable params: 21,002
Non-trainable params: 0

_________________________________________________________________
```

4.3 Practice exercises for using models:

▶ **Package pre-loading:**

[30]:
```python
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
```

▶ **Data pre-loading:**

[31]:
```python
df = pd.read_csv('ref6. Titanic.csv')

df.replace(False, 0, inplace=True)
df.replace(True, 1, inplace=True)

predictors = df.drop(['survived'], axis=1).to_numpy()
n_cols = predictors.shape[1]
target = to_categorical(df.survived)

pred_data = pd.read_csv('ref8. Titanic predictors data.csv')
pred_data.replace(False, 0, inplace=True)
pred_data.replace(True, 1, inplace=True)
```

▶ **Making predictions practice:**

[32]:
```python
# Specify, compile, and fit the model
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(n_cols, )))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(predictors, target)

# Calculate predictions: predictions
predictions = model.predict(pred_data)

# Calculate predicted probability of survival: predicted_prob_true
predicted_prob_true = predictions[:, 1]

# print predicted_prob_true
print(predicted_prob_true)
```

```
28/28 [==============================] - 0s 10ms/step - loss: 4.2202 - accuracy:
0.5785
[0.46402523 0.5921409  0.9999273  0.7663655  0.52398753 0.51178104
 0.28148016 0.5681087  0.4458243  0.9282887  0.5351253  0.53740656
 0.45552713 0.8895884  0.5160431  0.3519259  0.5469126  0.81220806
 0.43032184 0.8470085  0.9935336  0.5362614  0.28667322 0.58596545
 0.9431871  0.59334135 0.9650976  0.9699065  0.5989654  0.9669046
 0.5926729  0.8310348  0.55560493 0.54833597 0.57216716 0.99159616
 0.56204987 0.5336981  0.9594674  0.67796016 0.56222713 0.5251262
 0.71574086 0.5083044  0.576449   0.4511685  0.9595517  0.52266526
 0.8049595  0.9963193  0.89705867 0.26874813 0.7202186  0.92825663
 0.41592917 0.61183465 0.99998534 0.37461117 0.69608366 0.55560493
 0.3596782  0.43718696 0.40487978 0.9610486  0.5006672  0.44527555
 0.4620269  0.92955095 0.5375804  0.6937474  0.5351856  0.8505897
 0.37344947 0.43886235 0.56586283 0.60457915 0.5679306  0.5664585
 0.53257746 0.9858825  0.7241232  0.5036335  0.57002354 0.48111126
 0.53353465 0.77744293 0.48965743 0.81994593 0.62445176 0.7421769
 0.575239  ]
```