Building Deep Learning Models with Keras

LaTeX, Puteaux, 2020, 2021

Table of Contents

1 Creating a Keras model

1.1 [note-1] Model building steps

1.2 [code-1] Model specification

1.3 [quiz-1] Understanding the data

1.4 [task-1] Specifying a model

2 Compiling and fitting a model

2.1 [note-1] Why need to compile the model

2.2 [code-1] Compiling a model

2.3 [note-2] What is fitting a model

2.4 [code-2] Fitting a model

2.5 [task-1] Compiling the model

2.6 [task-2] Fitting the model

3 Classification models

3.1 [note-1] Classification

3.2 [note-2] Transforming to categorical

3.3 [code-1] Classification

3.4 [quiz-1] Understanding the classification data

3.5 [task-1] Last steps in classification models

4 Using models

4.1 [note-1] Using models

4.2 [code-1] Saving, reloading, and using the model

4.3 [code-2] Verifying model structure

4.4 [task-1] Making predictions

5 Requirements

1 Creating a Keras model

1.1 [note-1] Model building steps

- Specify Architecture
- Compile
- Fit
- Predict

1.2 [code-1] Model specification

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

predictors = np.loadtxt('../Datasets/1. 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 [quiz-1] 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]:
```
import pandas as pd
```

▶ **Data pre-loading**

[3]:
```python
df = pd.read_csv('../Datasets/2. Hourly wages.csv')
```

▶ **Quiz solution**

[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]:
```python
cols = df.columns
count = 0
for i in range(len(cols)):
    if ((len(df.iloc[:, i].unique()) == 2)
```

```
            and (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 [task-1] Specifying a model

▶ Package pre-loading

[7]:
```
import pandas as pd
```

▶ Data pre-loading

[8]:
```
df = pd.read_csv('../Datasets/2. Hourly wages.csv')

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

▶ Task practice

[9]:
```
# Import necessary modules
from tensorflow.keras.layers import Dense
from tensorflow.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 [note-1] Why need to compile the model

- Specify the optimizer:

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

2.2 [code-1] Compiling a model

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

predictors = np.loadtxt('../Datasets/1. 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 [note-2] What is fitting a model

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

2.4 [code-2] Fitting a model

[11]:
```
target = np.loadtxt('../Datasets/3. Hourly wages target data.csv',
                    delimiter=',')

model.fit(predictors, target)
```

```
17/17 [==============================] - 0s 2ms/step - loss: 74.1013
```

[11]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8c10086690>
```

2.5 [task-1] Compiling the model

▶ Package pre-loading

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

▶ Data pre-loading

[13]:
```
df = pd.read_csv('../Datasets/2. Hourly wages.csv')

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

▶ **Task practice**

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

2.6 [task-2] Fitting the model

▶ **Data pre-loading**

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

▶ **Task practice**

[16]:
```
# Import necessary modules
from tensorflow.keras.layers import Dense
from tensorflow.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 2ms/step - loss: 57.1017
```

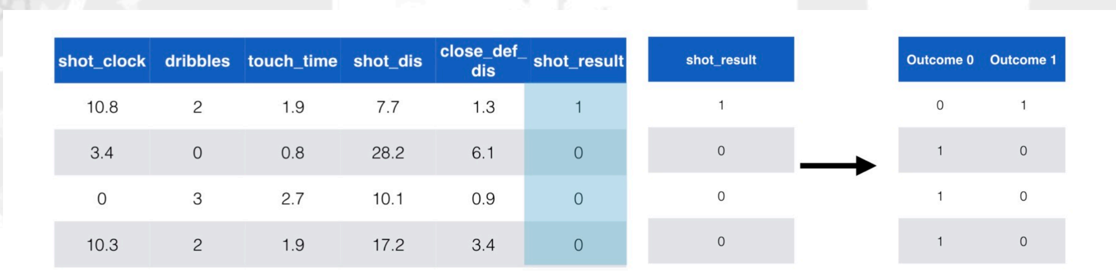
[16]: <tensorflow.python.keras.callbacks.History at 0x7f8be06e8c50>

3 Classification models

3.1 [note-1] Classification

- `'categorical_crossentropy'` loss function.
- Similar to log loss:
  - 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 [note-2] Transforming to 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-1] Classification

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


def data_preparation(data):
    df = data.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


data = pd.read_csv('../Datasets/4. Basketball shot log.csv')
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 [==============================] - 7s 2ms/step - loss: 0.6612 -
accuracy: 0.6086
```

[17]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8bcc2d2790>
```

3.4 [quiz-1] Understanding the classification data

- To start modeling with a new dataset for a classification problem. This data includes information about passengers on the Titanic. Use predictors such as `age`, `fare`, and where each passenger embarked from to predict who will survive. This data is from a tutorial on data science competitions. Look here for 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**

[18]:
```
import pandas as pd
```

▶ **Data pre-loading**

[19]:
```
df = pd.read_csv('../Datasets/5. Titanic.csv')
```

▶ **Quiz solution**

[20]:
```
df.head()
```

[20]:
```
   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
```

[21]:
```
df['age'].describe()
```

[21]:
```
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
```

[22]:
```
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 [task-1] Last steps in classification models

▶ **Package pre-loading**

[23]:
```
import pandas as pd
```

▶ **Data pre-loading**

[24]:
```
df = pd.read_csv('../Datasets/5. 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]
```

▶ **Task practice**

[25]:
```
# Import necessary modules
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.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 1ms/step - loss: 3.3554 - accuracy:
0.5791
```

[25]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8bcc1b3190>
```

4 Using models

4.1 [note-1] Using models

- Save.
- Reload.
- Make predictions.

4.2 [code-1] Saving, reloading, and using the model

[26]:

```
import pandas as pd
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import load_model


def data_preparation(data):
    df = data.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


def classification_model(n_cols):
    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'])
    return (model)


data = pd.read_csv('../Datasets/4. Basketball shot log.csv')
df = data_preparation(data)
predictors = df.drop(['shot_result'], axis=1).to_numpy()
n_cols = predictors.shape[1]
model = classification_model(n_cols)

model.save('../Models/1. Model of basketball shot log.h5')
my_model = load_model('../Models/1. Model of basketball shot log.h5')

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

4.3 [code-2] Verifying model structure

[27]:
```
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
_________________________________________________________________
```

4.4 [task-1] Making predictions

▶ **Package pre-loading**

[28]:
```python
import pandas as pd
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
```

▶ **Data pre-loading**

[29]:
```python
df = pd.read_csv('../Datasets/5. 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('../Datasets/6. Titanic predictors data.csv')
pred_data.replace(False, 0, inplace=True)
pred_data.replace(True, 1, inplace=True)
```

▶ **Task practice**

[30]:
```
# 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 846us/step - loss: 6.5769 -
accuracy: 0.5899
[0.6272365  0.69750106 0.99999297 0.6836752  0.6775551  0.6711255
 0.1392865  0.69602513 0.61362153 0.9100284  0.6842739  0.72476214
 0.6120106  0.9569266  0.673684   0.24523981 0.6843442  0.79242367
 0.46056715 0.8549943  0.98583883 0.68532735 0.14717013 0.68848526
 0.9899076  0.5917994  0.94522053 0.9758321  0.6220191  0.9549868
 0.6956421  0.87036264 0.7056431  0.6915692  0.70573115 0.98670435
 0.7000933  0.71821684 0.93538254 0.74775255 0.69923514 0.65353376
 0.7601951  0.56636417 0.709714   0.51969117 0.9882675  0.6146055
 0.75276166 0.9981142  0.959244   0.13130033 0.7325003  0.8988376
 0.6315635  0.6965266  0.9999964  0.58473164 0.6727381  0.7056431
 0.6269279  0.6300408  0.58141494 0.9906697  0.65385115 0.59693515
 0.60682535 0.8884945  0.72440106 0.6846784  0.68431187 0.8676819
 0.35615346 0.4988413  0.700127   0.69158924 0.7077357  0.70153165
 0.71755433 0.9637747  0.73155963 0.666571   0.6985411  0.6310478
 0.6829315  0.67343086 0.64595854 0.7937916  0.7421776  0.74876195
 0.5419165 ]
```

5 Requirements

[31]:
```
from platform import python_version
import tensorflow as tf

python_version = ('python=={}'.format(python_version()))
numpy_version = ('numpy=={}'.format(np.__version__))
tensorflow_version = ('tensorflow=={}'.format(tf.__version__))
pandas_version = ('pandas=={}'.format(pd.__version__))
```

```
writepath = '../../requirements.txt'
requirements = []
packages = [numpy_version, tensorflow_version, pandas_version]

try:
    with open(writepath, 'r+') as file:
        for line in file:
            requirements.append(line.strip('\n'))
except:
    with open(writepath, 'w+') as file:
        for line in file:
            requirements.append(line.strip('\n'))

with open(writepath, 'a+') as file:
    for package in packages:
        if package not in requirements:
            file.write(package + '\n')

max_characters = len(python_version)
for package in packages:
    if max(max_characters, len(package)) > max_characters:
        max_characters = max(max_characters, len(package))

print('#' * (max_characters + 8))
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * 2 + ' ' * 2 + python_version + ' ' *
      (max_characters - len(python_version) + 2) + '#' * 2)
for package in packages:
    print('#' * 2 + ' ' * 2 + package + ' ' *
          (max_characters - len(package) + 2) + '#' * 2)
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * (max_characters + 8))
```

```
#########################
##                     ##
##  python==3.7.9      ##
##  numpy==1.19.5      ##
##  tensorflow==2.4.1  ##
##  pandas==1.2.1      ##
##                     ##
#########################
```