In [54]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow import keras
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from keras.preprocessing.text import Tokenizer
```

In [55]:
```
data = pd.read_csv("/Users/zach/Downloads/SPAM text message 20170820 - Data.csv") #reads in data
data['Category'] = [1 if cat == "spam" else 0 for cat in data["Category"]] #changes labels to ints
data.head()
```

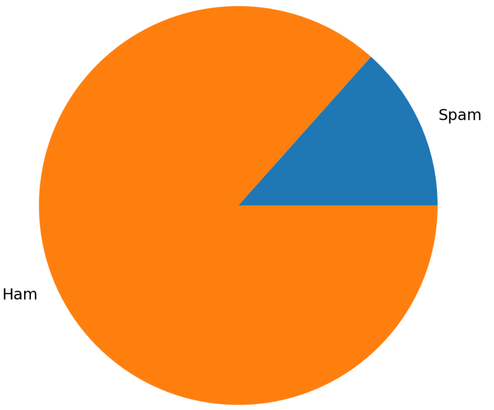
Out[55]:

	Category	Message
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

In [56]:
```
y = data['Category'] #sets y values
X = data['Message']  #sets X values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234) #splits to train an

labels = ['Spam', 'Ham'] #code to print graph
sizes = [747, 4825]
plt.figure(figsize=(20, 6), dpi=227)
plt.pie(sizes, labels = labels)
plt.show()
```



The dataset is a collection of real texts (labeled ham) and spam texts (labeled spam). The model should be able to predict if a given text is real or spam.

In [57]:
```
X = X.apply(lambda q : RegexpTokenizer('\w+').tokenize(q))
X.head(10)
```

Out[57]:
```
0    [Go, until, jurong, point, crazy, Available, o...
1                  [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, in, 2, a, wkly, comp, to, win, F...
3    [U, dun, say, so, early, hor, U, c, already, t...
4    [Nah, I, don, t, think, he, goes, to, usf, he,...
5    [FreeMsg, Hey, there, darling, it, s, been, 3,...
6    [Even, my, brother, is, not, like, to, speak, ...
7    [As, per, your, request, Melle, Melle, Oru, Mi...
8    [WINNER, As, a, valued, network, customer, you...
9    [Had, your, mobile, 11, months, or, more, U, R...
Name: Message, dtype: object
```

In [58]:
```
X = X.values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234, stratify=y) #splits
```

In [59]:
```
tokenizer = Tokenizer(num_words=20000)
tokenizer.fit_on_texts(X)
X = tokenizer.texts_to_matrix(X, mode='tfidf')
```

In [60]:
```
# build the model

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(20000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [61]:
```
# compile
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [62]:
```
_ = model.fit(X, y, batch_size=512, epochs=20, verbose=1, validation_split=0.2)
```

```
Epoch 1/20
9/9 [==============================] - 1s 46ms/step - loss: 0.6035 - accuracy: 0.8414 - val_loss: 0.4651 - val_
accuracy: 0.9803
Epoch 2/20
9/9 [==============================] - 0s 20ms/step - loss: 0.3771 - accuracy: 0.9877 - val_loss: 0.3127 - val_
accuracy: 0.9857
Epoch 3/20
9/9 [==============================] - 0s 21ms/step - loss: 0.2479 - accuracy: 0.9924 - val_loss: 0.2284 - val_
accuracy: 0.9857
Epoch 4/20
9/9 [==============================] - 0s 20ms/step - loss: 0.1725 - accuracy: 0.9948 - val_loss: 0.1757 - val_
accuracy: 0.9857
Epoch 5/20
9/9 [==============================] - 0s 20ms/step - loss: 0.1235 - accuracy: 0.9962 - val_loss: 0.1401 - val_
accuracy: 0.9865
Epoch 6/20
9/9 [==============================] - 0s 20ms/step - loss: 0.0897 - accuracy: 0.9969 - val_loss: 0.1160 - val_
accuracy: 0.9865
Epoch 7/20
9/9 [==============================] - 0s 25ms/step - loss: 0.0656 - accuracy: 0.9975 - val_loss: 0.0998 - val_
accuracy: 0.9865
Epoch 8/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0484 - accuracy: 0.9982 - val_loss: 0.0879 - val_
accuracy: 0.9865
Epoch 9/20
9/9 [==============================] - 0s 21ms/step - loss: 0.0360 - accuracy: 0.9989 - val_loss: 0.0823 - val_
accuracy: 0.9865
Epoch 10/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0271 - accuracy: 0.9993 - val_loss: 0.0763 - val_
accuracy: 0.9865
Epoch 11/20
9/9 [==============================] - 0s 21ms/step - loss: 0.0207 - accuracy: 0.9993 - val_loss: 0.0740 - val_
accuracy: 0.9865
Epoch 12/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0160 - accuracy: 0.9996 - val_loss: 0.0736 - val_
accuracy: 0.9857
Epoch 13/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0125 - accuracy: 0.9996 - val_loss: 0.0702 - val_
accuracy: 0.9857
Epoch 14/20
9/9 [==============================] - 0s 20ms/step - loss: 0.0099 - accuracy: 0.9996 - val_loss: 0.0708 - val_
accuracy: 0.9857
Epoch 15/20
9/9 [==============================] - 0s 21ms/step - loss: 0.0080 - accuracy: 0.9998 - val_loss: 0.0741 - val_
accuracy: 0.9848
Epoch 16/20
9/9 [==============================] - 0s 22ms/step - loss: 0.0065 - accuracy: 0.9998 - val_loss: 0.0731 - val_
accuracy: 0.9848
Epoch 17/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0054 - accuracy: 0.9998 - val_loss: 0.0760 - val_
accuracy: 0.9848
Epoch 18/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0045 - accuracy: 0.9998 - val_loss: 0.0787 - val_
accuracy: 0.9839
Epoch 19/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0038 - accuracy: 0.9998 - val_loss: 0.0804 - val_
accuracy: 0.9839
Epoch 20/20
9/9 [==============================] - 0s 19ms/step - loss: 0.0033 - accuracy: 0.9998 - val_loss: 0.0809 - val_
accuracy: 0.9839
```

With Sequential Model and 20 Epochs, Accuracy got to 99.96%

In [63]:
```
max_features = 10000
maxlen = 20000
batch_size = 32

model = models.Sequential()
model.add(layers.Embedding(max_features, 128, input_length=maxlen))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
```

In [64]:
```
# compile
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [65]:
```
_ = model.fit(X, y, batch_size=100, epochs=2, verbose=1, validation_split=0.2)
```

With CNN, we achieved 86.5% accuracy and around 40% loss. It also took longer to apply this method, about 5 mins per epoch. We could not even RNN with this dataset as it took way too long, on my machine over an hour an epoch.

In [66]:
```
model = models.Sequential()
model.add(layers.Embedding(max_features, 8, input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [67]:
```
# compile
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [68]:
```
_ = model.fit(X, y, batch_size=100, epochs=2, verbose=1, validation_split=0.2)
```

```
Epoch 1/2
45/45 [==============================] - 4s 71ms/step - loss: 0.4829 - accuracy: 0.8474 - val_loss: 0.3875 - va
l_accuracy: 0.8700
Epoch 2/2
45/45 [==============================] - 3s 66ms/step - loss: 0.4092 - accuracy: 0.8649 - val_loss: 0.3698 - va
l_accuracy: 0.8700
```

With embedding, we achieved 86.49% accuracy and loss around 50%. Executed much faster than CNN but not as fast as the sequential model.

Overall, I was happy with the accuracy and speed of the sequential model over anything else. The RNN would not even run under multiple hours, the CNN had low accuracy and ran slowly, and the embedding model ran faster than CNN but also had high loss and lower accuracy than the sequential model. I used Dr. Mazidi's implementations of each model, but had to fit them differently as my dataset was different.