In [1]:
```
import keras
import numpy as np
```

```
Using TensorFlow backend.
```

Keras

Keras is a neural network framework that wraps tensorflow (if you haven't heard of tensorflow it's another neural network framework) and makes it really simple to implement common neural networks. It's philosophy is to make simple things easy (but beware trying to implement uncommon, custom neural networks can be pretty challenging in Keras, for the purposes of this course you will never have to that though so don't worry about it). If you are ever confused during this homework, Keras has really good documentation, so just go to Keras Docs (https://keras.io)

Datasets

Keras has many datasets conviently builtin to the library. We can access them from the `keras.datasets` module. For this homework, we will be using their housing price dataset, their image classification dataset and their movie review sentiment dataset. To get a full list of their datasets, you can go to this link. Keras Datasets (https://keras.io/datasets). To use their datasets, we just import them and then call `load_data()`, load_data returns two tuples, the first one is training data, and the second one is testing data. See the example below

In [2]:
```
from keras.datasets import boston_housing
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```

You can also choose the proportion of training data you would like.

In [3]:
```
print("Size of training set before: ", x_train.shape)
(x_train, y_train), (x_test, y_test) = boston_housing.load_data(test_split=(
print("Size of training set after: ", x_train.shape)
```

```
Size of training set before:  (404, 13)
Size of training set after:  (455, 13)
```

In [4]:
```
from keras.utils import normalize
x_train = normalize(x_train, axis=1)
x_test = normalize(x_test, axis=1)
```

Models

Every thing in Keras starts out with a model. From an initial model, we can add layers, train the model on data, evaluate the model on test sets, etc. We initialize a model with `Sequential()`. Sequential refers to the fact that the model has a sequence of layers. Personally, I have very rarely used anything other than sequential, so I think its all you really need to worry about.

In [5]:
```
from keras.models import Sequential
model = Sequential()
```

Once we have a model, we can add layers to it with `model.add`. Keras has a really good range of layers we can use. For example, if we want a basic fully connected layer we can use `Dense`. I will now run through an example of using Keras to build and train a fully connected neural network for the purposes of regressing on housing prices for the dataset we loaded earlier.

In [6]:
```
from keras.layers import Dense
model.add(Dense(16, input_shape=(13,)))
```

This line of code adds a fully connected layer with 32 neurons. For the first layer of any model we always have to specify the input shape. In our case we will be training a fully connected network on the boston housing data, so each data point has 13 features. That's why we use an input_shape of (13,). The nice part about Keras is other than the input_shape for the first layer, we don't have to worry about shapes the rest of the time, Keras takes care of it. This can be really useful when you are doing complicated convolutions and things like that where working out the input shape to the next layer can be non-trivial.

Now let's add an Activation function to our network after our first fully connected layer.

In [7]:
```
from keras.layers import Activation
model.add(Activation('relu'))
```

Simple as that. We just added a relu activation to the whole layer. To see a list of activation functions available in Keras go to Keras Activations (https://keras.io/activations/). Now let's add the final layer in our model.

In [8]:
```
model.add(Dense(1))
```

Now we can use a handy utility in Keras to print out what our model looks like so far.

In [9]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                224
_________________________________________________________________
activation_1 (Activation)    (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 241.0
Trainable params: 241.0
Non-trainable params: 0.0
_________________________________________________________________
```

You can see it shows us what layers we have, the output shapes of each layer, and how many parameters there are for each layer. All this information can be really useful when trying to debug a model, or even for sharing your model architechture with others.

Training

Now for actually training the model. Before we train a model we have to compile it. `model.compile` is how you specify which optimizer to use and what loss function to use. Sometimes choosing the right optimizer can have a significant effect on model performance. For a list of optimizers look at Keras Optimizers (https://keras.io/optimizers). Choosing the right optimizer is mostly just trying each one to see which works better, there is some general advice for when to use each one but its basically just another hyperparameter. We also have to choose a loss function. Choosing the right loss function is really important since the loss function basically decides what the goal of the model is. Since we are doing regression we want to choose mean squared error, to get our output to be as close as possible to the label.

```
In [10]: model.compile(optimizer='SGD', loss='mean_squared_error')
```

Now we have to actually train our model on the data. This is really easy in Keras, in fact it only takes one line of code.

```
In [11]: model.fit(x_train, y_train, epochs=100)
```

```
Epoch 91/100
455/455 [==============================] - 0s - loss: 28.4114
Epoch 92/100
455/455 [==============================] - 0s - loss: 26.4436
Epoch 93/100
455/455 [==============================] - 0s - loss: 27.9273
Epoch 94/100
455/455 [==============================] - 0s - loss: 26.1374
Epoch 95/100
455/455 [==============================] - 0s - loss: 26.8396
Epoch 96/100
455/455 [==============================] - 0s - loss: 25.4413
Epoch 97/100
455/455 [==============================] - 0s - loss: 27.5721
Epoch 98/100
455/455 [==============================] - 0s - loss: 27.3682
Epoch 99/100
455/455 [==============================] - 0s - loss: 26.0724
Epoch 100/100
455/455 [==============================] - 0s - loss: 26.3213
```

Evaluation

Now that we have trained our model we can evaluate it on our testing set. It is also just one line of code.

In [12]:
```
print("Loss: ", model.evaluate(x_test, y_test, verbose=0))
```

```
Loss:  24.7450965433
```

This loss might seem very high and it is, mostly because there aren't very many training points in the dataset (also no effort was put into finding the best model).

We can also generate predictions for new data that we don't have labels for. Since we don't have new data, I will just demonstrate the idea with our testing data.

In [13]:
```
y_predicted = model.predict(x_test)
print(y_predicted)
```

```
 [ 15.82063103]
 [ 11.23418045]
 [ 12.74528122]
 [ 15.02112389]
 [ 12.74714279]
 [ 10.16311264]
 [ 13.76390266]
 [ 11.5139122 ]
 [  6.98116589]
 [  8.7193861 ]
 [  6.31607533]
 [  6.0158453 ]
 [ 14.71061325]
 [ 11.24947929]
 [ 14.57791519]
 [  8.02000237]
 [ 13.22076797]
 [ 11.93537521]
 [ 17.97393799]]
```

That's it. We have successfully (depending on your definition of success) built a fully connected neural network and trained that network on a dataset. Now its your turn.

Problem 1: Image Classification

We are going to build a convolutional neural network to predict image classes on CIFAR-10, a dataset of images of 10 different things (i.e. 10 classes). Things like aeroplanes, cars, deer, horses, etc.

(a) Load the cifar10 dataset from Keras. If you need a hint go to Keras Datasets (https://keras.io/datasets). This might take a little while to download.

In [40]:
```
from keras.datasets import cifar10
(cifar_x_train, cifar_y_train), (cifar_x_test, cifar_y_test) = cifar10.load_
```

(b) Initialize a Sequential model

In [41]:
```
cifar_model = Sequential()
```

(c) Add a `Conv2D` layer to the model. It should have 32 filters, a 5x5 kernel, and a 1x1 stride. The documentation here (https://keras.io/layers/convolutional/#conv2d) will be your friend for this problem. **Hint:** This is the first layer of the model so you have to specify the input shape. I recommend printing `cifar_x_train.shape`, to get an idea of what the shape of the data looks like. Then add a `relu` activation layer to the model.

In [42]:
```
from keras.layers.convolutional import Conv2D
print(cifar_x_train.shape)
cifar_model.add(Conv2D(32, (5, 5), padding='same', strides=(1, 1), input_sha
cifar_model.add(Activation('relu'))
```

```
(50000, 32, 32, 3)
```

(d) Add a `MaxPooling2D` layer to the model. The layer should have a 2x2 pool size. The documentation for Max Pooling is here (https://keras.io/layers/pooling/).

In [43]:
```
from keras.layers.pooling import MaxPooling2D
cifar_model.add(MaxPooling2D(pool_size=(2, 2)))
```

(e) Add another `Conv2D` identical to last one, then another `relu` activation, then another `MaxPooling2D` layer. **Hint:** You've already written this code

In [44]:
```
cifar_model.add(Conv2D(32, (5, 5), padding='same', strides=(1, 1)))
cifar_model.add(Activation('relu'))
cifar_model.add(MaxPooling2D(pool_size=(2, 2)))
```

(f) Add another `Conv2D` layer identical to the others except with 64 filters instead of 32. Add another `relu` activation layer.

In [45]:
```
cifar_model.add(Conv2D(64, (5, 5), padding='same', strides=(1, 1)))
cifar_model.add(Activation('relu'))
```

(g) Now we want to move from 2D data to 1D vectors for classification, to this we have to flatten the data. Keras has a layer for this called Flatten (https://keras.io/layers/core/#flatten). Then add a `Dense` (fully connected) layer with 64 neurons, a `relu` activation layer, another `Dense` layer with 10 neurons, and a `softmax` activation layer.

In [46]:
```
from keras.layers import Flatten
cifar_model.add(Flatten())
cifar_model.add(Dense(64))
cifar_model.add(Activation('relu'))
cifar_model.add(Dense(10))
cifar_model.add(Activation('softmax'))
```

Notice that we have constructed a network that takes in an image and outputs a vector of 10 numbers and then we take the softmax of these, which leaves us we a vector of 0s except 1 one and the location of this one in the vector corresponds to which class the network is predicting for that image. This is sort of the canonical way of doing image classification.

(h) Now print a summary of your network.

In [47]:
```
cifar_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 32, 32, 32)        2432      
_________________________________________________________________
activation_7 (Activation)    (None, 32, 32, 32)        0         
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 32)        25632     
_________________________________________________________________
activation_8 (Activation)    (None, 16, 16, 32)        0         
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 32)          0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 64)          51264     
_________________________________________________________________
activation_9 (Activation)    (None, 8, 8, 64)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 4096)              0         
_________________________________________________________________
dense_5 (Dense)              (None, 64)                262208    
_________________________________________________________________
activation_10 (Activation)   (None, 64)                0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                650       
_________________________________________________________________
activation_11 (Activation)   (None, 10)                0         
=================================================================
Total params: 342,186.0
Trainable params: 342,186.0
Non-trainable params: 0.0
_________________________________________________________________
```

(i) We need to convert our labels from integers to length 10 vectors with 9 zeros and 1 one, where the integer label is the index of the 1 in the vector. Luckily, Keras has a handy function to do this for us. Have a look here (https://keras.io/utils/#to_categorical)

In [67]:
```
from keras.utils import to_categorical
y_train_cat = to_categorical(cifar_y_train, 10)
y_test_cat = to_categorical(cifar_y_test, 10)
```

(j) Now compile the model with SGD optimizer and categorical_crossentropy loss function, also include `metrics=['accuracy']` as a parameter so we can see the accuracy of the model. Then train the model on the training data. For training we want to weight the classes in the loss function, so set the `class_weight` parameter of fit to be the `class_weights` dictionary. Be warned training can take forever, I trained on a cpu for 20 epochs (about 30 minutes) and only got 20% accuracy. For the purposes of this assignment you don't need to worry to much about accuracy, just train for at least 1 epoch.

In [68]:
```
cifar_model.compile(loss='categorical_crossentropy',
                    optimizer='SGD',
                    metrics=['accuracy'])
```

In [69]:
```
class_weights = {}
for i in range(10):
    class_weights[i] = 1. / np.where(cifar_y_train==i)[0].size

cifar_model.fit(cifar_x_train, y_train_cat, class_weight=class_weights, batc
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/1
50000/50000 [==============================] - 330s - loss: 0.0022 - acc:
0.1465 - val_loss: 10.6314 - val_acc: 0.1741
acc: 0.1295
0023 - acc: 0.1439
```

Out[69]:
```
<keras.callbacks.History at 0x11c88f8d0>
```

Now we can evaluate on our test set.

In [70]:
```
cifar_model.evaluate(cifar_x_test, y_test_cat)
```

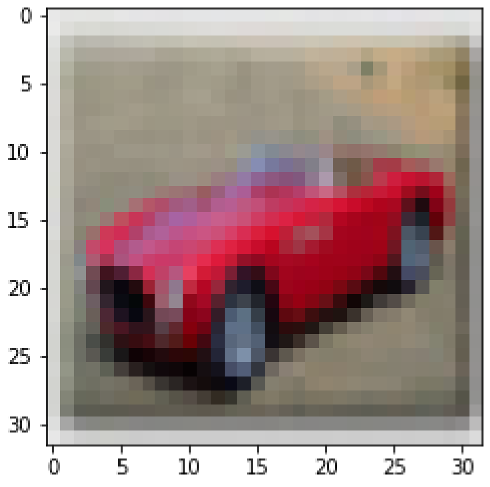
```
10000/10000 [==============================] - 22s      - ETA: 2s
```

Out[70]:
```
[10.63142777709961, 0.1741]
```

We can also get the class labels the network predicts on our test set and look at a few examples.

In [72]:
```
y_pred = cifar_model.predict(cifar_x_test)
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(cifar_x_test[1234])
print("Predicted label: ", np.argmax(y_pred[1234]))
print("True label: ", cifar_y_test[1234])
```

```
Predicted label:  9
True label:  [1]
```



Problem 2: Sentiment Classification

In this problem we will use Kera's imdb sentiment dataset. You will take in sequences of words and use an RNN to try to classify the sequences sentiment. First we have to process the data a little bit, so that we have fixed length sequences.

In [73]:
```
from keras.datasets import imdb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=1000, maxlen
```

In [74]:
```
def process_data(data):
    processed = np.zeros(len(data) * 200).reshape((len(data), 200))
    for i, seq in enumerate(data):
        if len(seq) < 200:
            processed[i] = np.array(seq + [0 for _ in range(200 - len(seq))
        else:
            processed[i] = np.array(seq)
    return processed
```

In [75]:
```
x_train_proc = process_data(x_train)
x_test_proc = process_data(x_test)
print(x_test_proc.shape)
```

```
(3913, 200)
```

The Embedding Layer is a little bit different from most of the layers, so we have provided that code for you. Basically, the 1000 means that we are using a vocabulary size of 1000, the 32 means we will have a vector of size 32 outputed, and the mask zero means that we don't care about 0, because we are using it for padding.

In [89]:
```
imdb_model = Sequential()
```

In [90]:
```
from keras.layers.embeddings import Embedding
from keras.layers import LSTM
imdb_model.add(Embedding(1000, 32, input_length=200, mask_zero=True))
```

(a) For this problem, I won't walk you everything like I did in the last one. What you need to do is as follows. Add an LSTM layer with 32 outputs, then a Dense layer with 16 neurons, then a relu activation, then a dense layer with 1 neuron, then a sigmoid activation. Then you should print out the model summary.

In [91]:
```
imdb_model.add(LSTM(32))
imdb_model.add(Dense(16))
imdb_model.add(Activation('relu'))
imdb_model.add(Dense(1))
imdb_model.add(Activation('sigmoid'))
```

(b) Now compile the model with binary cross entropy, and the adam optimizer. Also include accuracy as a metric in the compile. Then train the model on the processed data (no need to worry about class weights this time)

In [95]:
```
imdb_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

After training we can evaluate our model on the test set.

In [96]:
```
print("Accuracy: ", imdb_model.evaluate(x_test_proc, y_test)[1])
```

```
3913/3913 [==============================] - 18s
Accuracy:  0.50396115517
```

Now we can look at our predictions and the sentences they correspond to.

In [97]:
```
y_pred = imdb_model.predict(x_test_proc)
```

In [98]:
```
y_pred = np.vectorize(lambda x: int(x >= 0.5))(y_pred)
correct = []
incorrect = []
for i, pred in enumerate(y_pred):
    if y_test[i] == pred:
        correct.append(i)
    else:
        incorrect.append(i)
word_dict = inv_map = {v: k for k, v in imdb.get_word_index().items()}

print(list(map(lambda x: word_dict[int(x)] if x != 0 else None, x_test[corre
```

```
Downloading data from https://s3.amazonaws.com/text-datasets/imdb_word_in
dex.json (https://s3.amazonaws.com/text-datasets/imdb_word_index.json)
['the', 'great', 'kids', 'in', 'own', 'as', 'ever', 'is', 'lack', 'of',
 'great', 'like', 'situation', 'do', 'japanese', 'more', 'he', 'time', 'a
gain', 'br', 'any', 'he', 'british', 'great', 'like', 'fantastic', 'lif
e', 'plot', 'rating', 'by', 'rock', 'in', 'had', 'expected', 'things', 't
o', 'political', 'clearly', 'in', 'as', 'ever', 'in', 'and', 'that', 'a
n', 'br', 'of', 'films', 'he', 'doubt', 'most', 'from', 'one', 'her', 'pl
ot', 'and', 'from', 'could', 'and', 'not', 'from', 'he', 'doubt', 'and',
 'really', 'it', 'so', 'and', '10', 'in', 'can', 'are', 'fast', 'hilariou
s', 'other', 'but', 'of', 'its', 'lot', 'of', 'love', 'it', 'otherwise',
 'and', 'and', 'there', 'will', 'and', 'br', 'so', 'and', 'but', 'and',
 'actors', 'years', 'play', 'rock', 'in', 'own', 'as', 'found', 'of', 'an
d', 'would', 'and', 'every', 'book', 'but', 'and', 'and', 'like', 'effect
s', 'and', 'and', 'her', 'box', 'other', 'if', 'mind', 'in', 'can', 'is',
 'ever', 'for', 'well', 'one', 'is', 'got', 'fun', 'they', 'of', 'and',
 'br', 'that', 'to', 'his', 'at', 'and', 'film', 'and', 'and', 'and', 'i
t', 'is', 'and', 'fun', 'really', 'it', 'other', 'his', 'life', 'is', 'qu
ite', 'are', 'fast', 'hilarious', 'what', 'see', 'because', 'worth', 'aft
er', 'and', 'of', 'and', 'br', 'as', 'ever']
```

After making this I realized that keras' method for converting from word index back to words is broken right now (see this open github issue (https://github.com/fchollet/keras/issues/5912)). So we can't actually see what the sentences look like.