Programming 4

Submit Assignment

Due Friday by 11:59pm **Points** 100 **Submitting** a file upload
File Types zip and py **Available** until Dec 12 at 11:59pm

the COMS W4701 - Artificial Intelligence - Programming Homework 4

Image Classification using Convolution Neural Networks

Due: 11:59pm on Friday, December 8th
Total Points: 100

You are welcome to discuss the problems with other students but you must turn in your own work. Please review the academic honesty policy for this course (at the end of the syllabus page).

As a reminder, any assignments submitted late will incur a 20 point penalty. No submissions will be accepted later than 4 days after the submission deadline.

The instructions below are fairly specific and it is okay to deviate from implementation details. **However: You will be graded based on the functionality of each function/method. Make sure the function signatures (names, parameter and return types/data structures) match exactly the description in this assignment.**

Introduction

In this assignment you will use the Keras Neural Network API for Python to build neural networks for image classification.

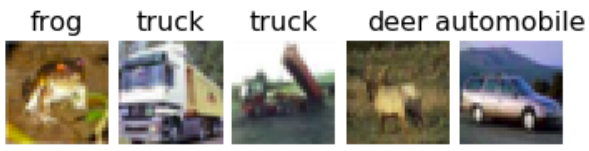
Data Set

We will work on the CIFAR-10 image data set mentioned in class and described here: **https://www.cs.toronto.edu/~kriz/cifar.html** **(https://www.cs.toronto.edu/~kriz/cifar.html)**

The data set contains 60.000 images labeled with 10 different categories:

Numeric ID	Category Name
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

Each image is 32x32 pixels large and there are three color channels (red, green blue). Each image can therefore be represented as three 32x32 matrices or one 32x32x3 cube.
Here are some example images:



Getting started: Installing Numpy, Keras and TensorFlow

Keras is a high-level Python API that allows you to easily construct, train, and apply neural networks. However, Keras is not a neural network library itself and depends on one of several neural network backends. We will use the Tensorflow backend. TensorFlow is an open-source library for neural networks (and other mathematical models based on sequences of matrix and tensor computations), originally developed by Google.

Numpy is a numeric computing package for Python. Keras uses numpy data structures.

Installing Numpy, TensorFlow and Keras:

We suggest that you use the Python package management system pip.
On most systems, the following commands will work:

```
$ pip install numpy matplotlib
$ pip install tensorflow
$ pip install keras
```

Note that this will likely install the CPU version of TensorFlow that does not use the GPU to speed up neural network training. For this assignment, training on the CPU will be sufficient, but if your computer has a GPU (or you want to try running the assignment in the cloud), follow the installation instructions on the tensorflow page.

If you get stuck during the installation, you can find installation instructions for each package here:

Tensorflow: https://www.tensorflow.org/install/ (https://www.tensorflow.org/install/)
Keras: https://keras.io/#installation (https://keras.io/#installation)

Testing your Setup:
To test your setup, run a Python interpreter and type the following:

```
$ python
Python 3.6.1 |Anaconda 4.4.0 (x86_64)| (default, May 11 2017, 13:04:09)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
RuntimeWarning: compiletime version 3.5 of module 'tensorflow.python.framework.fast_tensor_util' does not match run
time version 3.6
 return f(*args, **kwds)
```

```
>>> import keras
Using TensorFlow backend.
>>>
```

You can ignore any warnings printed by TensorFlow.

Using an interactive Python shell

For this homework assignment, it can be useful to use an interactive Python interpreter to run experiments. We recommend using Jupyter Notebook, which is a web-application that allows you to type and run Python code in a web browser. Other options include IPython qtconsole or the built-in ipython console in Anaconda's Spyder IDE.

If you write your code directly in the interpreter, make sure to copy-paste it back into the **homework4.py** file that you will submit to Courseworks.

Part 1 (20 pts) - Loading the CIFAR-10 Data

Download the template file **homework4.py.**

The following code fragment imports the CIFAR-10 data using Keras.

```
from keras.datasets import cifar10
train, test = cifar10.load_data()
xtrain, ytrain = train
xtest, ytest = test
```

xtrain, ytrain, xtest, and ytest are **numpy n-dimension arrays (https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ndarray.html)**, containing the training and testing data.
You can look at the format of these arrays:

```
>>> xtrain.shape
(50000, 32, 32, 3)
>>>  ytrain.shape
(50000, 1)
```

The input training data (xtrain) is a 4-dimensional array containing 50000 images, each of them a 32x32x3 tensor. Numpy arrays can be indexed like nested Python lists, so xtrain[0] will give you the first 32x32x3 image.

The input label (ytrain) is a vector containing the numeric class for each image (see table above for what the numeric IDs mean). For example, xtrain[0] is am image of a frog and therefore ytrain[0] contains the value 6.

Visualizing Images

If you want to take a look at the individual images, you can do so using matplotlib (this step is optional).

```
from matplotlib import pyplot as plt
%matplotlib inline
xtrain, ytrain_1hot, xtest, ytest_1hot = load_cifar10()
plt.imshow(xtrain[6])
```

1-hot representation for class labels

The output layer of the neural networks we will train will contain 10 neurons corresponding to the 10 classes.

The classifier predicts the class whose corresponding neuron has the highest activation. We need to convert the numeric indices for each image into a 1-hot vector of length 10, so that the for class label n the n-th element is 1 and all other elements are 0. For example, if the class label is 6, we should get the 1-hot vector [0 0 0 0 0 0 1 0 0 0].
The class labels for the entire training and set should then be represented as a 50000x10 matrix, and the testing data as a 10000x10 matrix (there are 10k test images).

Write the function *load_cifar10()*, which should load the cifar-10 data as described above and should return 4 numpy arrays *xtrain, ytrain_1hot, xtest, ytest_1hot.* Your function should convert the y arrays into the 1-hot representation. You can either do this using loops (slow), using numpy fancy indexing (see numpy documentation), or by using appropriate functions in the numpy or Keras API.

Your function should also do the following **normalization** on the data. The R,G and B values for each pixel range between 0 and 255. Before returning the training data, normalize it so that these value range between 0.0 and 1.0.

Part 2 (20 pts) - Multilayer-Neural Network

Designing the Network

We will start by building a Neural Network with a single hidden layer. This network will be easy to train, but will not have good performance. Keras neural networks are created in two steps. First, you specify the layers of the network (the "computation graph"), then you compile this network so you can train the weights and evaluate it. This split is typical of most neural network packages.

You will complete the function *build_multilayer_nn()*, that creates and returns a Keras model object, but will not train it.

Each neural network consists of a number of layers. In class, we talked about layers as vectors (or in the convolution case 2-dimensional arrays) of neurons. In Keras, the layers are more general. You can think of each-layer as a function applied to an n-dimensional array and resulting in another m-dimensional array.

The model itself is an instance of the class **keras.Sequential (https://keras.io/models/sequential/)**.

```
nn = Sequential()
```

You can then add layers to this object one-by-one.

The single hidden layer we will use is an instance of class **keras.layers.Dense (https://keras.io/layers/core/#Dense)**. A dense layer is one in which all neurons are connected to all inputs to the layer (i.e. a typical neural network layer). The following creates a layer with 100 neurons and using the recitifier function as the activation function.

```
hidden = Dense(units=100, activation="relu")
nn.add(hidden)
```

The output layer, as mentioned above, will be a dense layer containing 10 neurons. As activation function, we will use the Softmax function, which adjusts the activations at the output layer so that activations sum up to 1.0. It is then possible to think of the output activation as a "probability distribution" indicating the probability that the input belongs to a certain class.

```
output = Dense(units=10, activation="softmax")
nn.add(output)
```

When defining the first layer of the network, which is the hidden layer, we need to also define the shape of the input array to that layer. For all other layers, the input shape is infered automatically.

The problem is that Dense layers only accept 1 dimensional input, but each image is a 32x32x3 array. We could convert each 32x32x3 array into an array of size 3072 and then specify the input shape of the hidden layer as

```
hidden = Dense(units=100, activation="relu", input_shape=(3072,))
```

However, it would be nicer if our network would take the image data as input directly. So instead of preprocesing the data, we can pass the input through a layer of type **keras.layers.Flatten (https://keras.io/layers/core/#flatten)** , which simply squashes the 32x32x3 array into a vector of length 3072. Research this layer type in the Keras documentation and add it to the model definition before the hidden layer.

When you run the function *build_multilayer_nn()* it will return the Keras model object. You can run the method summary() on this object to print out a summary of the network topology.

```
>>> nn = build_multilayer_nn()
>>> nn.summary()
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
flatten_1 (Flatten) (None, 3072) 0
_________________________________________________________________
dense_1 (Dense) (None, 100) 307300
_________________________________________________________________
dense_2 (Dense) (None, 10) 1010
=================================================================
Total params: 308,310
Trainable params: 308,310
Non-trainable params: 0
_________________________________________________________________
```

Training the Model

The function *train_multilayer_nn(model, xtrain, ytrain)* has already been written for you.

```
def train_multilayer_nn(model, xtrain, ytrain):
    sgd = optimizers.SGD(lr=0.01)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    model.fit(xtrain, ytrain_1hot, epochs=20, batch_size=32)
```

Before the training process, the model needs to be configured ("compiled") by providing a loss function, optimizer algorithm, and a metric to be measured. The loss function we will use is categorical crossentropy, which essentially measures how different the output distribution is from the target distribution (i.e. the one-hot target vector).
The optimizer is stochastic gradient descent (SGD). The idea behind SGD is similar to the gradient descent optimization methods discussed in class, but the training samples are shuffled in every training epoch and the gradient is updated with respect to a batch of training samples rather than a single one. We use a learning rate of 0.01, set the batch size to 32 and train for 20 epochs.

If you set up the model topology correctly, you should be able to train the model like this:

```
>>> xtrain, ytrain_1hot, xtest, ytest_1hot = load_cifar10()
>>> nn = build_multilayer_nn()
>>> train_multilayer_nn(nn, xtrain, ytrain_1hot)
```

The training should not take more than 5 minutes and should report a final accuracy of about 0.52 on the training set. The model is not yet fitted well to the training data, so training has not yet converged. You can continue running the training process for more epochs, but model performance will improve only slowly.

Evaluating the Trained Model

Once the model has been trained, you should evaluate it on the test set.

```
>>> nn.evaluate(xtest, ytest_1hot)
```

The result should be around 50%. This is not terrible considering the baseline for this problem is 10% (since there are 10 classes), but about half the images are still misclassified. We will now build a better model that converges after fewer iterations (though training will take longer).

Important: Copy/paste the output of evaluate() as a comment at the beginning of your build_multilayer_nn() method!

Part 3 (30 pts)- Convolution Neural Network

As discussed in class, a convolution neural network consists of convolution layers and pooling layers for feature extraction, and dense layers for classification.

- You can create a convolution layer using the keras.layers.Conv2D layer type.

```
Conv2D(32, (3, 3), activation='relu', padding="same")
```

will use filters of size 3x3 to create 32 feature maps of size 32x32. The output shape will be 32x32x32.

- You can create a pooling layer using the keras.layers.MaxPooling2D layer type.

```
MaxPooling2D(pool_size=(2, 2))
```

will take the feature maps created by a convolution layer as input and scale down their size. The pool_size parameters specify the *factor* by which the image maps are sampled down. Applying this layer to the 32x32x32 representation output by the convolution layer will result in a 16x16x32 representation.

To make the model generalize well to the test data, some drop-out is useful. Drop-out randomly sets some units to 0 during each training update, which prevents the model from over-fitting.

- You can create a dropout layer using the keras.layers.Dropout layer type.

```
Dropout(0.25)
```

will create a layer that drops 25% of the units in each training step.

Designing the Network

Write the function build_convolution_nn() that constructs a convolution neural network containing of:

- **Two** convolution layers, consisting of 32 feature maps with a filter size of 3x3.

- One pooling layer, that reduces the size of each feature map to 16x16.
- One drop-out layer that drops 25% of the units.
- Two more convolution layers, consisting of 32 feature maps with a filter size of 3x3.
- Another pooling layer, that reduces the size of each feature map to 8x8. The output shape should be 8x8x32
- One drop-out layer that drops 50% of the units.
- Feed this output into a regular multilayer neural network with two hidden layers of size 250 and 100. The output layer should be of size 10, as before.

Note that the first convolution layer will have to specify the input shape as 32x32x3. You will have to flatten the output of the feature extraction stage before you feed it into the dense layers.

It may be useful to consult the **Keras documentation on convolution layers (https://keras.io/layers/convolutional/)** .

Training the model

Write the function *train_convolution_nn(model, xtrain, ytrain)*, that trains the model using same parameters we used for the multilayer neural network. The two functions will be identical at this point.

Run the training and evaluation process as before. Training will take much longer (20 epochs take about 20 minutes on my laptop) and the initial accuracy on the training data will be low. By the end of the 20 epochs, the model will be fitted to the training data pretty well (around 85%). Without the drop-out layers, the optimizer would be able to fit the model almost perfectly to the training data, but this would be a result of overfitting (i.e. you would not get as good a result on test).

(If you have access to a machine with an Nvidia GPU and you have the GPU version of TensorFlow, training will be much faster).

On the test set, this model should provide accuracies in the high 60s. This is not great (the state of the art on this task is about 96%), but a significant achievement, considering that the baseline is 10%.

Optimizing the model

Experiment by modifying the network topology (adding/removing layers etc.) or training parameters (learning rate, number of epochs, batch size, ...). To get full credit on this part, you should get an accuracy of > 70% on the test data.

Important: Copy/paste the best output of evaluate() as a comment at the beginning of your build_convolution_nn() method!

Part 4 (30 pts) - Convolution Neural Network for Binary Classification

The 10 categories in the CIFAR-10 data can be grouped into 2 super-categories: animals and vehicles. Your task is to modify the model above so that it performs binary classification: Is the output an animal (1) or a vehicle (0).

You need to change the following:

1. Write the function get_binary_cifar10() that returns the arrays xtrain, ytrain, xtest, ytest as before, but now ytrain should be a binary vector of size (50000,) and ytest should be a vector of size (10000,), where 1 indicates an animal and 0 indicates a vehicle.

2. Write the function build_binary_classifier() that creates the structure of a convolution neural network that performs binary classification. The output to this network should be a single neuron (a dense layer of size 1) with sigmoid activation.
3. Write a function train_binary_classifier() that trains the model on the new output data produces by get_binary_cifar10(). Use binary_crossentropy instead of categorical_crossentropy.
4. Evaluate the model. Answer the following questions in a comment at the beginning of the file: Is the binary classification task easier or more difficult than classification into 10 categories? Justify your response.

   Important: Copy/paste the best output of .evaluate() as a comment at the beginning of your build_convolution_nn() method!

What You Need to Submit

For this homework assignment, you need to submit the filled-in template file **homework4.py**. You do not have to submit any trained models.

We will not be able to run all of your code and retrain your neural networks, so we will go by the performance you report in comments. **You must report the actual results produced by your neural networks.** If we find that your results are unrealistically good or bad given the changes you made, we will run your code to verify. You will lose points for inaccurately reported or missing results.