Programming Assignment #3

The **coding component** of this assignment should be **submitted through e-Learning** before **3:59 pm Central Standard Time on Tuesday, December 4, 2018**. The **report component** of this assignment is due **due at the start of class** on **Tuesday, December 4, 2018**. These deadlines are without exceptions unless permission was obtained from the instructor **in advance**.
You may collaborate with other students, discuss the problems and work through solutions together. However, you **must write up your code and solutions on your own**, without copying another student's work or letting another student copy your work. In your solution for each problem, you must write down the names of any person with whom you discussed it. This will **not** affect your grade.

Problem: We return to the problem of **hand-written digit recognition** that we explored in HW#2. Recall that, in HW#2, we performed a two-step process of dimensionality reduction (with PCA) followed by 10-class classification (with SVMs). The goal here is to develop a simple deep network to perform 10-class classification directly on this data set. We will seek to understand the effect of various parameters on the complexity of the deep network as well as the final prediction outcome.

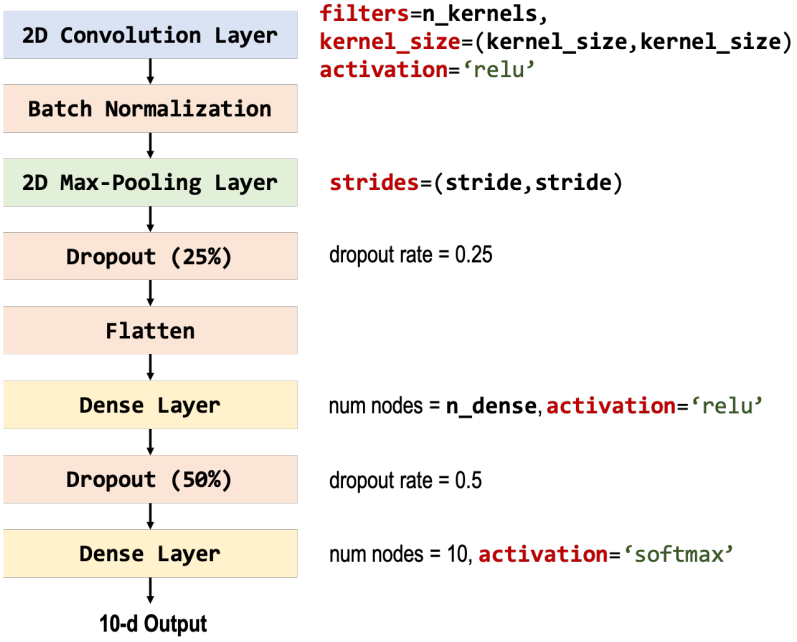

Figure 1: Architecture of the deep network we will build and train. Note that all activation functions are set to RelU. This network has four parameters that we will experiment with: `n_kernels`, `kernel_size`, `stride`, `n_dense`. See the Keras Documentation (https://keras.io/) for details on how to implement these layers.

Data Set: We will use the USPS digits data set. This data set has already been pre-processed and partitioned into train (5266 digits), validation (1094 digits) and test (930 digits). Note that

- the train, validation and test data are each *tensors* of size $(n \times 16 \times 16 \times 1)$, where n is the respective number of digits (examples). The greyscale images are represented as a $16 \times 16 \times 1$ tensor in order to maintain consistency for matrix multiplication. If the images were RGB, then the images would be represented as a $16 \times 16 \times 3$ tensor.
- each label is represented as a *one-hot vector*. That is, since this is a 10-dimensional problem, we represent the label as a 10-dimensional vector, with entries all zero except for the one corresponding to the label. For instance, the digit 3 is represented as $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]^T$.
- Load the data set using `pickle`:

```
import pickle
usps_data = pickle.load(open('usps.pickle', 'rb'))
```

Keras and TensorFlow: Keras[1] is a high-level API for neural networks and deep learning and supports several

[1] https://keras.io/

different open-source machine learning frameworks including TensorFlow[2], CNTK and Theano. For the purpose of this homework, we will use TensorFlow to design, train and test deep networks. Note that TensorFlow is only compatible with Python 3.6.

Modeling: Write a function that returns the architecture of a deep model constructed using Keras,

```
build_model(n_kernels=8, kernel_size=3, stride=2, n_dense=32)}
```

to build a `Sequential()` model. The four arguments to `build_model` are parameters that define the behavior of various layers in the network. The architecture we will use is shown in Figure 1. For the 2D convolutional layer, the parameter `n_kernels` specifies the number of kernels (filters), and each kernel is a square of size `kernel_size`. The parameter `stride` defines the stride of the max-pooling layer as a `stride` $\times$ `stride` pooling window. The parameter `n_dense` defines the number of hidden nodes in a densely connected layer.

Training: Keras supports numerous loss functions and optimizers. For this experiment, we will **compile the deep model** using the **categorical cross-entropy** loss function and the **Adam** optimizer with learning rate $\eta = 10^{-4}$. You can fit a deep model to the training data with:

```
from keras.callbacks import LearningRateScheduler
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
history = model.fit(data['x']['trn'], data['y']['trn'],
                    epochs=2, batch_size=16, verbose=2,
                    validation_data=(data['x']['val'], data['y']['val']),
                    callbacks=[annealer])
```

Parameter Selection in Deep Learning: Execute following experiments and write a brief report answering the following questions:

a. (**The effect of Filter Size**, 25 points) Keeping all the other parameters fixed to their default values, train models with `n_kernels` $= 1, 2, 4, 8, 16$, and evaluate them on the test set. Plot (i) `n_kernels` vs. training and test error, and (ii) `n_kernels` vs. total number of network parameters. What seems to be a reasonable choice of `n_kernels`?

b. (**The effect of Kernel Size**, 25 points) Keeping all the other parameters fixed to their default values, train models with `kernel_size` $= 1, 2, 3, 4, 5$, and evaluate them on the test set. Plot (i) `kernel_size` vs. training and test error, and (ii) `kernel_size` vs. total number of network parameters. What seems to be a reasonable choice of `kernel_size`?

c. (**The effect of Stride**, 25 points) Keeping all the other parameters fixed to their default values, train models with `stride` $= 1, 2, 3, 4$. Plot (i) `stride` vs. training and test error, and (ii) `stride` vs. total number of network parameters, and evaluate them on the test set. What seems to be a reasonable choice of `stride`?

d. (**The effect of Dense Layer Size**, 25 points) Keeping all the other parameters fixed to their default values, train models with `n_dense` $= 16, 32, 64, 128$. Plot (i) `n_dense` vs. training and test error, and (ii) `n_dense` vs. total number of network parameters, and evaluate them on the test set. What seems to be a reasonable choice of `n_dense`?

Upload: Make sure all your code is in a single file. Also ensure that your code can be it can be executed by calling the main function. Upload your file through e-Learning before the deadline.

For Fun: If you'd like to further explore deep learning on other data sets, the package `keras.datasets`[3] contains several data sets for image classification, movie review sentiment classification, news topics classification and a bigger handwritten digit database (MNIST). Many deep learning algorithms also require a considerable amount of **data pre-processing and augmentation**. Data augmentation, especially, is the process of increasing the data set size from an existing data set. This is achieved through a combination of shifts, rotations, zooms, clips, flips and other operations to "generate" new data. Keras has a neat package, `keras.preprocessing.ImageDataGenerator`[4], that will generate batches of tensor images in real-time.

[2] https://www.tensorflow.org/
[3] https://keras.io/datasets/
[4] https://keras.io/preprocessing/image/