Assignment 4

March 29, 2017

1 Foundations of Data Mining: Assignment 4

Please complete all assignments in this notebook. You should submit this notebook, as well as a PDF version (See File > Download as).

```
In [1]: %matplotlib inline
        from preamble import *
        plt.rcParams['savefig.dpi'] = 100 # This controls the size of your figures
        # Comment out and restart notebook if you only want the last output of each
        InteractiveShell.ast_node_interactivity = "all"
```

1.1 Backpropagation (3 points)

Figure 1 illustrates a simple neural network model.

It has single input x, two layers with one neuron each. The activation function of both layers is ReLU.

The parameters w_0 and w_1 (no biases) are initialized to the following values $w_0 = 1$ and $w_1 = 2$. Implement a single update step of the gradient descent algorithm by hand. Run the update state for the following two data points:

- $(1, 2)$
- $(2, 3)$

The goal is to model the relationship between two continuous variables. The learning rate is set to 0.1

Provide the solution in the following format:

- A choice for a loss function
- Compute graph for training the neural network
- Partial derivative expression for each of the parameters in the model
- The update expression for each of the parameters for each of the data-points
- The final value of both parameters after the single step in the gradient descent algorithm

1.2 Training Deep Models (3 points)

The model in the example code below performs poorly as its depth increases. Train this model on the MNIST digit detection task.

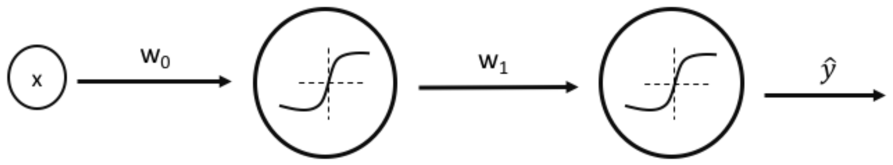


Figure 1

Examine its training performance by gradually increasing its depth: - Set the depth to 1 hidden layer - Set the depth to 2 hidden layers - Set the depth to 3 hidden layers

Modify the model such that you improve its performance when its depth increases. Train the new model again for the different depths: - Set the depth to 1 hidden layer - Set the depth to 2 hidden layers - Set the depth to 3 hidden layers

Submit an explanation for the limitation of the original model. Explain your modification. Submit your code and 6 plots (can be overlaid) for the training performance of both models with different depths.

```
In [ ]: # (You don't need to change this part of the code)
        from __future__ import print_function
        import numpy as np
        np.random.seed(1234)

        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers.core import Dense, Dropout, Activation
        from keras.optimizers import SGD
        from keras.utils import np_utils


        import matplotlib.pyplot as plt

        batch_size = 128
        nb_classes = 10
        nb_epoch = 10
```

```
In [ ]: # (You don't need to change this part of the code)
        # the data, shuffled and split between train and test sets
        (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
```

```
In [ ]: # Use this parameter to change the depth of the model
        number_hidden_layers = 1  # Number of hidden layers
```

```
In [ ]: # Model
        model = Sequential()
        model.add(Dense(512, input_shape=(784,), activation='sigmoid'))
        model.add(Dropout(0.2))

        while number_hidden_layers > 1:
            model.add(Dense(512))
            model.add(Activation('sigmoid'))
            model.add(Dropout(0.2))
            number_hidden_layers -= 1


        model.add(Dense(10))
        model.add(Activation('softmax'))

        model.summary()

        model.compile(loss='categorical_crossentropy',
                      optimizer=SGD(),
                      metrics=['accuracy'])
```

```
In [ ]: # Training (You don't need to change this part of the code)
        history = model.fit(X_train, Y_train,
                            batch_size=batch_size, nb_epoch=nb_epoch,
                            verbose=1, validation_data=(X_test, Y_test))
        score = model.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        # list all data in history
```

```python
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

1.3 Convolutional Neural Networks for Filtering (2 points)

Convolutional neural networks are well suited for analyzing images. They can be used to apply various image filtering operations.

The goal of this exercise is to design a CNN model that applies 2 filters to its input images. The input images are 128x128 RGB color images, encoded as 128x128x3 tensor with floating point value normalized between 0 and 1. The RGB format is such that the pixels address by: [:, :, 0] encode the red pixels of the image, the pixels addressed by [:, :, 1] define the green pixels and pixels addressed by [:, :, 2] define the blue pixels.

Design a convolutional neural network that will: 1. Apply the sepia filter to the image 2. Apply Gaussian smoothing to the image

Use the specification of the sepia and the Gaussian filter below.

You answer should contain: - The definition of the architecture of the CNN - Number of layers - Number of filters per layer - Shape of the filter per layer - Values of each of the parameters of the CNN when using a 5x5 Gaussian smoothing filter - The dimensions of the output image when a 5x5 Gaussian smothing is applied

The sepia effect gives warmth and a feel of vintage to photographs. The sepia filter is defined as:

$$R_o = (R_i * .393) + (G_i * .769) + (B_i * .189) G_o = (R_i * .349) + (G_i * .686) + (B_i * .168) B_o = (R_i * .272) + (G_i * .534) + (B_i * .131) \tag{1}$$

Gaussian blurring is an effect that reduces the noise and details in an image. Gaussian smoothing filter:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- A discretized version of the filter is given by the following table:

1
4
7
4
1
4
16
26
16
4

7
26
41
26
7
4
16
26
16
4
1
4
7
4
1

- To normalize the filter response, each value should divided by 273. This is a truncated discretized Gaussian filter with a σ of 1.

1.4 Model Design (2 Points)

Various decisions need to be made in a modeling process to address specific properties of the data and the modeling goal. In this task, you are given a description of a data structure and a goal for which you need to design a model.

Produce a figure depicting your model. Briefly explain the figure and justify all decisions made in the modeling process. In detail, describe at least: - Input data format - Number of layers - Type of layers (Dense, Recurrent, Convolutional - 1D, 2D, 3D) - Regularization - Model output - Loss function

The training and execution procedures for the model may differ, so you can use different descriptions for both.

Data and goal description:

The goal of this task is to generate captions for short video clips.

The video data is structured as sequences of color images. The model needs to be able to process a number of consecutive images that form a short video clip. The training data consists of video clips (few seconds) and a short caption (5-10 words).

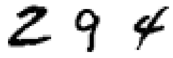
For simplicity, the accuracy of the model is evaluated on the exact prediction of the caption. In other words, the model needs to produce correctly the specific words in a specific order for each video.

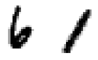
1.5 MNIST Calculator (5 points)

During the lectures you have seen a CNN model that can be successfully trained to classify the MNIST images. You have also seen how a RNN model that can be trained to implement addition of two numbers.

Using the KERAS (or TensorFlow) library, design and train a model that can learn how to add numbers directly from the MNIST image data. More specifically, the model should input a sequence of a set of images and produces a cumulative sum of the numbers represented by the digits in the images.

For example:

Input 1: 

Input 2: 

Output: 355