Practical Introduction to Neural Networks
Homework 3
Due: May 22, 2019

Overview

Before starting with this homework please make sure that you have TensorFlow and Jupyter installed on your laptop. In addition, you will need a **GPU** for training CNNs. If you don't have available GPU, you could use the Compute Engine of Google Cloud Platform. Please check the setup guide on canvas. For each problem in this homework we have provided a Jupyter notebook template with which you can get started.

Templates are available at the course Github Repository:
`https://github.com/shlizee/PracticalIntroductionNN`

Please submit Jupyter notebooks with your code (and outputs) by the due date to your Github repository and place a link to your repository in the canvas assignment.

Problem 1: Visualizing and Interpreting CNNs

In this problem, you will start from a convolutional neural network model (VGG-16) which has been pre-trained to perform image classification on the ImageNet dataset. You will use this pre-trained model for the following tasks. Before you start, please download the **checkpoint** file from website `https://github.com/tensorflow/models/tree/master/research/slim` or by command line

```
wget http://download.tensorflow.org/models/vgg_16_2016_08_28.tar.gz
```

For details, please follow the instructions in **hw3_1_template.ipynb**

Part a: Visualizing Feature Maps

To get familiar with the architecture of VGG-16, use the pre-trained model to visualize the feature maps at different layers.

Part b: Saliency Maps

In this part you will implement a saliency map for the input image. Saliency map is representation of an image into a more meaningful and easier to analyze signal. For CNNs saliency maps can be used as a way to tell which part of the image influenced the classification decision made by the network.

Part c: Fooling Images

In this part, you will experiment with perturbations of an input image for which the pre-trained network is sensitive. Some such perturbations appear very similar to humans, but will cause the perturbed image to be incorrectly classified by the pre-trained network.

Figure 1: example of style transfer

Part d: Class Visualization

In this part, you are asked to synthesize an image to maximize the classification score of a particular class. This will provide some intuition on what the network is looking for when it classifies images of that class.

Problem 2: Neural style transfer

In this part you will implement the style transfer technique form "Image Style Transfer Using Convolutional Neural Networks" (Gatys et al., CVPR 2015). The general idea of the algorithm is to take two images, and produce a new image that reflects the content of one but the artistic "style" of the other. To accomplish that you will first formulate a loss function that matches the content and style of each respective image in the feature space of a CNN, and then perform a gradient descent optimization on the pixels of the generated image. The CNN that you will be using for the pupose of style transfer is the VGG-16 pre-trained on ImageNet (same as problem 1). For details, please check the instructions in **hw3_2_template.ipynb**.

Part a: Definition and Computing the Loss

Follow the instructions on template to define and compute the loss for adequate for neural style transfer.

Part b: Generate Artistic Images

Run the three pairs of images provided with the template. In addition, to the given images, you use **your own content image** with **UW style image** to generate a creative and cool image in **Husky** style.

Part c: Feature Inversion

In previous parts the image to be generated was initiated by the content image. In this part, you are asked to initiate the image to be generated as a random image (3 channels random pixel values) instead of the content image. The goal is to use the optimization of the content loss (style weight $= 0$) and reconstruct the content source image. If the optimization is set up correctly, you will end up with something that looks like the original content image.

Problem 3: Stock Price Prediction

In this problem, you will implement a Recurrent Neural Network to predict stock prices. You will predict the stock prices of Google, Tesla, and Dow Jones Industrial Average Index, respectively. For details, check the instructions in **hw3_3_template.ipynb**

Part a: Data Preprocessing

Datasets are given in csv files. Use the tools **pandas** and **sklearn** to retrieve and normalize the data between 0 and 1. Split your training, validation and testing set wisely. You will only use **Open** price as your input (i.e. input dimension = 1).

Part b: Implement an RNN model

Implement a many-to-one Recurrent Neural Network (i.e. given certain number of days data, predict the stock price of the next day.) You are free to use RNN, GRU, or LSTM (or compare between) and any number of layers and architecture. In your testing, plot the ground truth and your predicted values for 100 days.

Problem 4: Character-level text generation

You will implement a multi-layer Recurrent Neural Network (RNN, LSTM, and GRU) for training/sampling from character-level language models, which takes one text file as input and trains an RNN that learns to predict the next character in a sequence. The RNN can then be used to generate text character by character that will look like the original training data.

Part a: Text Loader

Create a class to preprocess the text file data (**shakespeare.txt**) for later usage. For details, check the instructions in **text_utils.py**.

Part b: RNN model

Create a class that implements the character level Recurrent Neural Network model. For details, check the instructions in **char_rnn_model.py**

Part c: Training and Sampling

Train the model and use it to generate new text. For details, check the instructions in **hw3_4_template.ipynb**.