Introduction to Deep Learning
Assignment 1: building multi-layer perceptrons (MLPs), convolutional neural networks (CNNs) with Tensorflow/Keras.

October 2024

Introduction

In this assignment, you are going to use TensorFlow/Keras deep learning library to build various neural network models for classification and regression tasks on image datasets.

You are advised to use the following textbook which contains a large variety of practical examples of the Keras API as a reference.

Aurelien Geron: Hands-on Machine Learning with Scikit-learn, Keras and Tensorflow, Edition #2

The key objectives of this assignment:

- Learn how to define and train simple neural networks in Keras.
- Gain practical experience by comparing various MLP and CNN architectures, getting some intuitions for manually tuning hyperparameters and their effects on model performance.
- Apply this knowledge to develop a more complex CNN for the "tell-the-time" problem.

Note: This assignment is going to be heavy in terms of computational requirements, therefore you are recommended to find a suitable platform with GPU acceleration to train your models. We recommend the following options:

1. Google Colab/DeepNote with TPU/GPU acceleration. To change your runtime type on Colab go to *Runtime → Change runtime type → Select 'TPU' or 'GPU' option.*
2. Using the computers in the lab rooms (most of them have a 2GB Nvidia 1050GTX card which should be sufficient for smaller models.) You can check out the *General information → Instructions* on Brightspace on how to connect to the lab computers remotely.
3. If you have a laptop with an NVidia GPU or a high-end CPU you can potentially complete everything on your own device. PyTorch setup instructions will differ based on your OS/system (instructions)

Task 1: Learn the basics of Keras API for TensorFlow.

Start with reading the section "Implementing MLPs with Keras" from *Chapter 10 of Geron's textbook (pages 295-320).* Then install TensorFlow 2.0+ and experiment with the code included in this section (Brightspace *General information → Instructions*). Additionally, study the official documentation `https://keras.io/` and get an idea of the numerous options offered by Keras (layers, loss functions, metrics, optimizers, activations, initializers, regularizers). Don't get overwhelmed with the number of options – you will frequently return to this site in the coming months. Your tasks:

1. Check out this official repository with many examples of Keras implementations of various sorts of deep neural networks here. We recommend cloning this repository and trying to get some of these examples running on your system (or Colab/DeepNote). In particular, experiment with *mnist_mlp.py* and *mnist_cnn.py* scripts which show you how to build simple neural networks for the MNIST dataset (useful for the next task).
2. Next, take the two well-known datasets: Fashion MNIST (introduced in Ch 10, p. 298) and CIFAR-10. The first dataset contains 2D (grayscale) images of size 28x28, split into 10 categories; 60,000 images for training and 10,000 for testing, while the latter contains 32x32x3 RGB images (50,000/10,000 train/test). Apply two reference networks on the fashion MNIST dataset:

(a) A **multi-layer perceptron** described in detail in *Ch. 10, pp. 299-307*

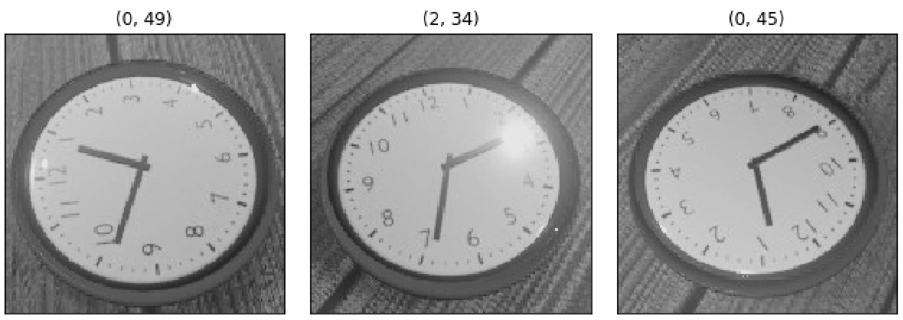
(b) After lecture 6/7: a **convolutional neural network** described in *Ch. 14, p. 447.*

Experiment with both networks, trying various options: initializations, activations, optimizers (and their hyperparameters), regularizations (L1, L2, Dropout, no Dropout). You may also experiment with changing the architecture of both networks: adding/removing layers, number of convolutional filters, their sizes, etc. For optimizing your hyperparameters you should use a validation set (10% of the training set). After you have found the best-performing hyperparameter sets, take the 3 best ones and train new models on the CIFAR-10 dataset to see whether your performance gains translate to a different dataset. Provide your thoughts on these results in the report (e.g. what are the main reasons for the difference in performance?).

The purpose of this task is NOT to get the highest accuracy. Instead, you are supposed to gain some practical experience with tuning networks, running multiple experiments (with the help of some scripting), and, very importantly, documenting your findings and understanding reasonable ranges for your hyperparameters. In your report, you have to provide a concise description of your experiments, results, and conclusions. The quality of your work (code and report) is more important than the quantity or the accuracy you've achieved.

Task 2: Develop a "Tell-the-time" network.

We have produced a big collection of images of an analog clock along with labels corresponding to the time shown in these images. Your task is to develop and train a CNN that would tell the time as correctly as possible. Here is a random sample of 3 images from our collection with the corresponding time (hour, minutes) displayed on top of these images:



The dataset can be downloaded from here (150x150 images) or here (75x75 images) and it consists of 18000 grayscale images (18000x150x150 or 18000x75x75) contained in *'images.npy'*. The labels for each sample are represented by two integers (18000x2, *'labels.npy'* file), that correspond to the hour and minute displayed by the clock. You can see that each image is rendered from a different angle and rotation and they might contain light reflections from within the scene making this a non-trivial problem. For your experiments, we suggest splitting your data into 80/10/10% splits for training/validation and test sets respectively. Remember to shuffle your dataset as the sample files are ordered. We suggest using the smaller dataset for your initial tests and runs (75x75 images) and then reporting your results on the larger (150x150) dataset.

Subtasks:

1. The problem of correctly telling the time can be formulated either as a multi-class classification problem (for example, with 12x60=720 classes representing each minute label) or a regression problem (for example, predicting the number of minutes after 12 o'clock). Therefore, your goal is to come up with different representations for the labels of your data adapt the output layer of your neural network and see how it impacts the training time and performance. No matter which architecture and loss function you will use when reporting results also provide "common sense" accuracy: the absolute value of the time difference between the predicted and the actual time (e.g., the "common sense" difference between "predicted" 11:55 and the "target" 0:05 is just 10 minutes and not 11 hours and 50 minutes!). Minimizing this "common sense" error measure is the main objective of this assignment! Notice that it is a common situation in Machine Learning: we often train models using one error measure (e.g., cross-entropy loss) while the actual performance measure that we are interested in is different, e.g., the accuracy (the percentage of correctly classified cases). Here are some ideas that you should experiment with when building your models:

(a) **Classification** - treat this as a n-class classification problem. We suggest starting out with a smaller number of categories e.g. grouping all the samples that are between $[3:00 - 3:30]$ into a single category (results in 24 categories in total), and trying to train a CNN model. Once you have found a working architecture, increase the number of categories by using smaller intervals for grouping samples to increase the 'common sense accuracy'. Can you train a network using all 720 different labels? What problems does such a label representation have?

(b) **Regression** - try to build a network that predicts the time using a single output node in the following format: $[\text{"}03:00\text{"} \rightarrow y = 3.0]$; $[\text{"}05:30\text{"} \rightarrow y = 5.5]$, where categorical labels of hours and minutes get transformed to a single continuous value. What kind of loss function would you need to use for such a task? What kind of problems does such a representation have?

(c) **Multi-head models**: your neural network can have multiple output heads e.g. one head for predicting hours and another head for predicting minutes. These heads could also potentially have different losses and targets (e.g. multi-class for hours and regression for minutes). Implement such a model and see how the performance of such a model compares to the previous models. How would you explain the differences?

Note: more information on how to build multi-head models can be found in Ch 10, p. 308-310 of the textbook.

(d) **Label transformation (optional)**: think how the labels could be reformulated by using periodic functions (e.g. sine and cosine functions to represent the angles on the unit circle). How would you adapt your neural network to these kinds of labels? How many output nodes would be sufficient and what activation function would be perfect for this task? Note: you can achieve the lowest 'common sense error' by using this approach.

2. Use the knowledge gained by working with other datasets in the previous parts of this assignment to optimize your final models and decrease the error of telling the time as much as possible (common sense error of below 10 minutes is achievable using relatively simple CNN architectures). You should also compare the different ways of representing your labels and different neural network output layer combinations. You should use an 80:20% ratio for the train/test sets respectively. Document your experiments and findings in the report.

Deliverables: Your submission should consist of a report in *pdf* format (at most 10 pages) and neatly organized code (Jupyter notebooks) so that we can reproduce your results. You also need to submit .py files of all the notebooks (by converting *.ipynb* to *.py*). You can find more information about report writing on the Brightspace $\rightarrow$ Assignments section.