Machine Learning - Homework 4

Neural Network Approaches for Classifying Images

Carlos Correia - 76512
Maxwell Júnior - 79457

Full code for the report available on Google Colab - https://colab.research.google.com/drive/19icsx9WFoFp0jUPMqjh0xy9o0AZZhI54?usp=sharing

MNIST Dataset with images representing numbers from 0-9 will be the base for our model creation/evaluation approaches, resolving different parameters, aiming for an accurate model that given blank images with numbers, can state the class for new blank images w/numbers (0-9).

Image data is defined by a tensor with the shape *(nr_of_images) * (image width) * (image height) * (nr_or_channels - colors)* Number of channels is usually 3 as we're using the RGB color system. This dataset has only a color channel, sufficient to represent numbers to train the model. Number of images will be 70000. 60K for Train and 10K for Test. Image width will be 28x28 pixels.

Preprocessing done involves only representing a color from an interval [0,1] by dividing the sequencial pixel color data by 255 (2^8, 8 bits to represent an RBG component). A model is firstly compiled having as base the dimensions of the dataset to create the initial layer-layer-neuron weights (W) and respective bias (B) matrixes.

The last hidden layer of all the observed networks will be a flatten layer that merges the image 3d data (color, width and height properties) into a single result so that it can then normally be fed to the usual hidden layer(s) (called Dense in the used framework) for a feed-forward base neural network. (using the *softmax* activation function so that the output layer can be fed using the categorical cross-entropy loss function - that can assign values that sum up to one regarding the distribution of the class labels)

We're now ready to train the compiled model, in other words, fitting the model with the training data. A batch size of 256 will be used so that an update of the gradient descent is only done after concatenating 256 examples (to improve performance of the learning but decreasing the ability to learn - *underfitting*)

Gradient descent is used, arranging examples in sets of 256 (60000*0.8/256 ~ 188 examples for each epoch) . Then a high number of epochs is designated to maximize accuracy, but using a strategy called patience, which will record the highest accuracy at each step and stop the epoch iterations after x rounds of patience are passed without improvement of the former.

The accuracy that is recorded follows from the validation set (20% of the training data, taken from the sample without replacement, on contrary of the training accuracy, which if used, will lead to a model that has no ability to generalize - overfitting)

If the accuracy (against the validation set) improves without having to train the whole training dataset for each epoch then the remaining examples will be discard and the next epoch starts. This can be observed in the output of the .fit() function of the Keras framework.

```
Epoch 205/10000
175/188 [==========================>...] - ETA: 0s - loss: 0.2905 - accuracy: 0.9192
Epoch 00205: val_accuracy improved from 0.91633 to 0.91658, saving model to mnist_baseline_best.h5
188/188 [==============================] - 1s 3ms/step - loss: 0.2898 - accuracy: 0.9192 - val_loss: 0.3068 - val_accuracy: 0.9166
```

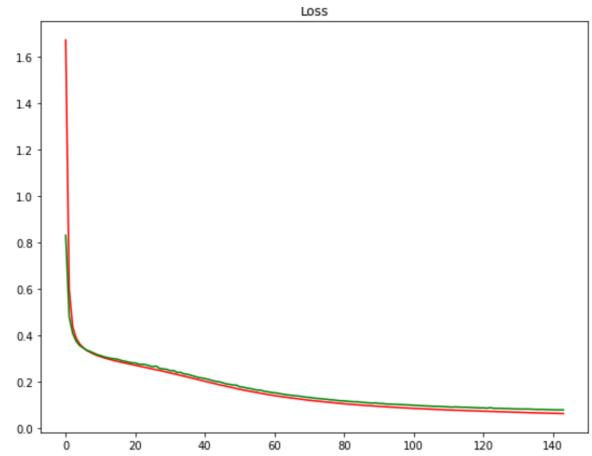
If x patience more epochs don't improve the accuracy of the model then early stop callback is called and the relevant data to plot the model evolution in terms of loss/accuracy against the number epochs is collected.

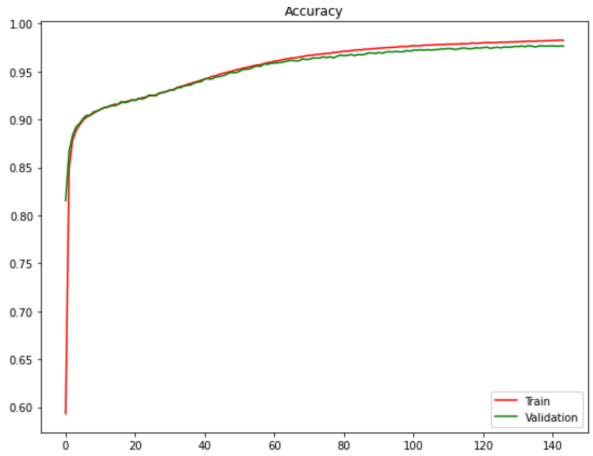
The default values from the introductory class were not tweaked but studied and laid down on this report, due to the observation of high values of accuracy / low loss on validation for learning and even higher numbers of accuracy for the test set (also sampled from the original dataset without replacement). Numbers are 0.9166 for validation set and 0.9203 for the test set, making the feed-forward neural networks a very promising tool for evaluating imagery data.

Convolution Network Approach - These “evolution” of the neural network in terms of hidden layers will aid the image classification problem by having an heuristic in consideration, regarding the “shape” of potential images to come, having special interest when combined with other learning models that can learn the heuristic on a different part of the cortex / computer system. The heuristic data is considered to be a *GlorotUniform* Instance (samples from an Uniform distribution) - https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotUniform

This heuristic is applied on the weights from the very beginning of the network, upon receiving the image tensor, so called regularisation. A window with the size of the kernel matrix (generated with the uniform *KernelInitializer* from samples of the input tensor) will be sled on the image resulting on the convolutions instead of normal matrix multiplications (dot product between kernel matrix cells and input image channel).

Using a window of 4pixels of width/height with strides and padding as the first hidden layer and a second hidden layer with the max pooling strategy (instead of the sliding window with the kernel heuristic we just max out contiguous pixels defined by the pool_size arg, in this case 2 was used). Afterwards, use the same remaining layers to be able to train the images and you get improved results, resulting in an accuracy of almost 97% on the test set.





Also, pseudo-random computation can remove some neurons before the last layers of the net (flatten and cross-entropy output) in a technique called holdout. With 50% holdout, training accuracy decreases. Validation accuracy might improve in contrast to no holdout, usually by chance. This is noticeable by looking at the bottom charts from the 0th epoch onwards. This might be explained by the randomised removal of neurons in the core of the network, removing also a big proportion of neurons that would overfit the data because of the accuracy measurement on sampling with replacement technique (for the training data)