Machine Learning class (IELE4014): Homework 3 Report

ELKIN CRUZ, UNAL

1 INTRODUCTION AND DATASET

This document documents the stuff done to try to solve a problem of artificial vision (or machine learning on vision).

The dataset used was NORB-small[1]. The problem posed by NORB-small consists in identify an object in an image from one of five categories. All images have a size of 96×96 pixels and have two channels, the images are binocular images, i.e., two cameras, side by side, were used for a single shot. The five categories of objects are: four-legged animals, human figures, airplanes, trucks, and cars.

The NORB-small dataset has 23400 images for training and 23400 images for testing. I broke the testing set into two parts, each of the same size. The first 17700 images were used in validation and the last 17700 were used in testing. Remember that for a binary classification problem we can use the Chernoff equation to estimate the margin of the testing error from the "real" error (which we could get if we had infinite datapoints). With 17700 datapoints and a confidence of 95%, we know that the margin would be aproximately 1.4%. Unfortunatelly the same calculations get very tricky for the case of multiple classes, and therefore I could not give any certainty on the results obtained here.

To solve the problem, I decided to use Neural Networks. A simple Neural Network (Multilayer-Perceptron with one hidden layer) was used as "baseline", its accuracy was 70.4%. Other models, such as Convolutional Networks were also tested.

The source code used to run the experiments can be found at https://github.com/helq/ml_hw3.

2 OPTIMIZATION/TRAINING ALGORITHM

All Neural Networks were implemented on Python using the Tensorflow library. All models were trained using the Adam algorithm with parameters: learning rate of 0.001, β_1 of 0.9, β_2 of 0.999, ϵ of 1×10^{-8}, and no locking. All values, except for the learning rate, were the standard values setted by default by the library.

Dropout was only used for one of the models (for the biggest model, based on a model that has 98% accuracy on the MNIST dataset). The loss function used is the cross entropy between the output of the networks and the hot vectors associated to the real vector value.

The batch size for training was setted to 128. This value was selected, because it was the preferred in many examples of convolutional network training.

3 MODELS

A total of 6 models were tested to try solve this problem. The first two model can be considered as baseline, given their simplicity. The others involve modifications to two convolutional networks.[2]

All models have the same inputs and outputs. The inputs are the 96×96 (times two) pixels on the binocular images. The outputs are six real numbers, in inference the output with the biggest value tells the class.

3.1 One-layer Perceptron

The first model can be seen as a generalization of linear regression, where 5 linear regressions are performed simultaneously each one representing the likelihood that one image belongs to a class or another. The activation function for this perceptron is lineal.

3.2 Two-layer Perceptron

The second model is a perceptron with one hidden layer. The hidden layer has 900 neurons with activations tanh.

3.3 MNIST-Convolutional adaptation

This model is an adaptation from the example code for convolutional networks in Tensorflow Examples[3]. The original code is teted on on the MNIST dataset and it gets a very high testing accuracy, aproximately 98% accuracy.

The Network is defined by five layers C1, S2, C3, S4 and FC5. C1 is a convolutional layer with input a binocular image (96×96 twice), 32 filters and 5×5 kernels. S2 is a Max Pooling layer with kernel of 2 and strides of 2. C3 is a convolutional layer with 64 filters and a kernel size of 3. S4 is a max pooling layer with kernel 2 and strides of 2. FC5 is a fully connected layer (percetron) that takes the $24 \times 24 \times 64$ outputs from S4. FC5 outputs a value for each one of the 5 classes.

This model was tested with two different activation functions, tanh and relu.

3.4 LeCun's model

This model is more complicated than MNIST-adaptation (above), but it is smaller on the number of weights. This is the model defined in the original paper NORB was introduced LeCun, Huang, and Bottou (2004), they argue (in the paper) that this model can get up to 6% error testing.

The Network is defined by six layers: C1, S2, C3, S4, C5, and FC6. C1 is a layer composed of 3 convolutional layers with kernels of size 5, the first two layers take a monocular image (left or right) and output 2 filters, and the third layer takes the binocular image and outputs 4 filters. S2 is a max pooling layer with kernel of size 4 and strides of 1. C3 is a layer composed of 24 distinct convolutional layers with kernels of size 3, each convolutional layer takes 4 filters

[1] NORB-small can be found at http://www.cs.nyu.edu/~ylclab/data/norb-v1.0-small/

[2] A small note about the current state of libraries for Deep Learning. I tried using more complicated pre-implemented models such as Mobilenet (Howard et al. 2017). I thought that given these implementations are well tested, it should be better to use them than to implemented from scratch, because I could add many bugs to them. To my surprise, this proved to be harder than implementing a model from scratch. Tensorflow is a big library and it is in constant development and improvement, it is also a little "low level", and constantly new libraries/wrappers appear to make some tasks easier (e.g., creating layers for NNs, or log training process). This has incentivated the creation of multiple wrappers around Tensorflow, each one with its own idioms and ways to do their stuff. I failed completely trying to make use of one of such wrappers and couldn't try more complex models therefore :(.

[3] https://github.com/aymericdamien/TensorFlow-Examples/blob/d43c58c948cb4fec189b13a39a620e54879a3495/examples/3_NeuralNetworks/convolutional_network.py

("images") from S2, one from the first monocular filter, other from the second monocular filter, and two more from the 4 binocular filters. S4 is a max pooling layer with kernel of size 3 and strides of 1. C5 is a convolutional layer that takes all 24 filters and outputs 80 filters in total, with kernels of size 6. FC6 is a fully connected layer that takes all outputs from the 80 filters and outputs 5 values for each one of the classes.

This model was tested with two different activation functions, tanh and relu.

4 TRAINING AND ANALYSIS

The results of training, validation and testing for each model can be seen next.

4.1 One-layer Perceptron

This simple model (basically 5 unbalanced linear regression problems, each one classifying one of the classes) is able to capture some information about the images, its testing accuracy was 61.9%, which is significantly better than 20% if the the classification was random.

In Figure 1, the validation accuracy is shown. The perceptron goes from not being able to distinguish at step 50 (each step is an execution of Adam in a batch of size 128) to (apparently) an accuracy measure of aprox. 60%-70% after 1000 iterations.

Figure 2 shows the change of the loss function as time goes. The loss on training is smaller than in validation, as expected, but it never reaches zero, which means that the model cannot memorize the training data.

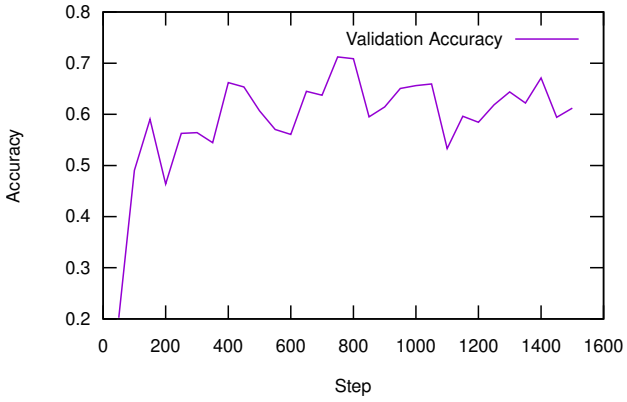

Fig. 1. Validation Accuracy for One-layer Perceptron

4.2 Two-layer Perceptron

This model is a little more complex than the one-layer percetron, therefore it is able to memomorize a little better as it can be seen in Figure 4. Its testing accuracy was 70.4%, better indeed than one-layer (linear regression). The best model would probably be the model at step 2100, because it seems that at around 2100 the loss is smallest and more consistent than in 3000.

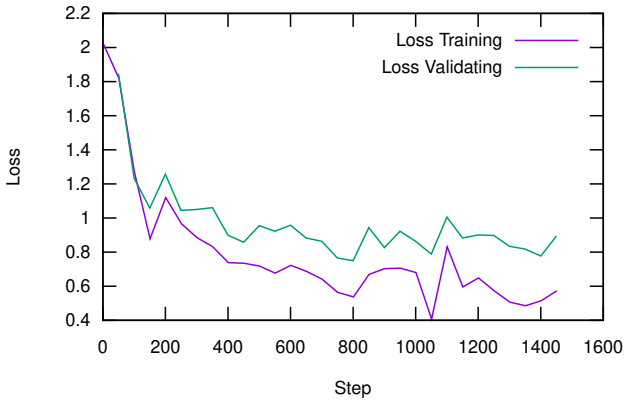

Fig. 2. Loss value function for One-layer Perceptron

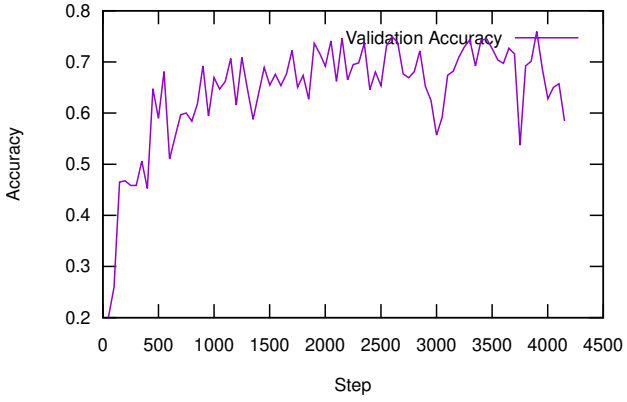

Fig. 3. Validation Accuracy for Two-layer Perceptron

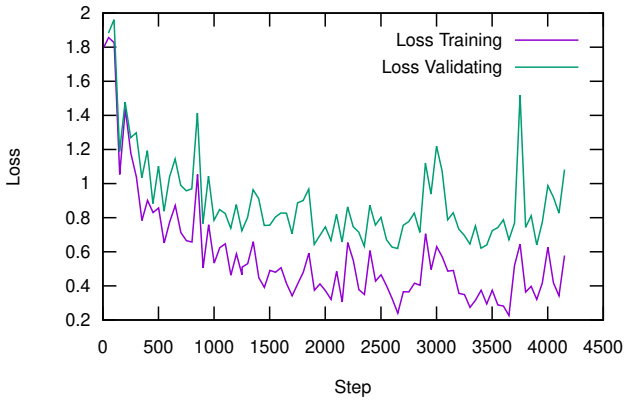

Fig. 4. Loss value function for Two-layer Perceptron

4.3 MNIST-Convolutional adaptation

There are two variations of this model, one using tanh as activation function and other using relu. As it can be seen in Figure 5, the model with activation tanh has a little problem when it is left too long optimizing. The testing accuracy for the model tanh was 86.9% while the model relu was 90.4%.

In Figure 6, we can see that we arrive at a model that has memorized the training data, and as time goes on, it seems that the model is overfitting (the loss function increases for the validation set), probably a good model has been already found with 500 steps.

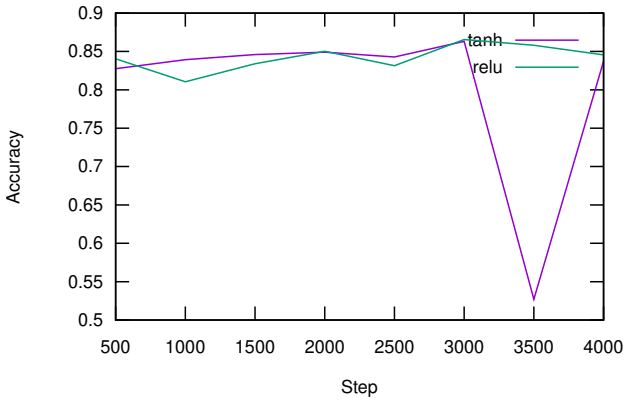

Fig. 5. Validation Accuracy for MNIST adapted

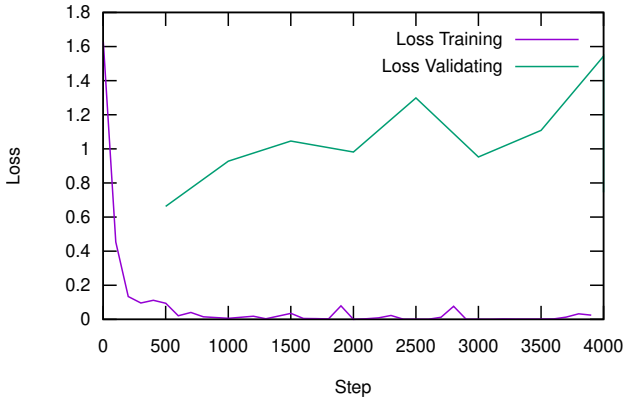

Fig. 6. Loss value function for MNIST adapted (with relu activation)

4.4 LeCun's model

This model was proposed by LeCun, Huang, and Bottou (2004) when they introduced the NORB dataset. I hoped to reproduce the same results they got, namely, around 7% error score (about 93% accuracy), but the final result was a little disappointing. I got 90.1% accuracy on the testing set. The validation accuracy on training can be seen in Figure 7.

The model I trained probably doesn't performs as well as it does in the original paper because, probably, I'm using different training and initialization algorithms, and other little details on implementations could be different from the paper.

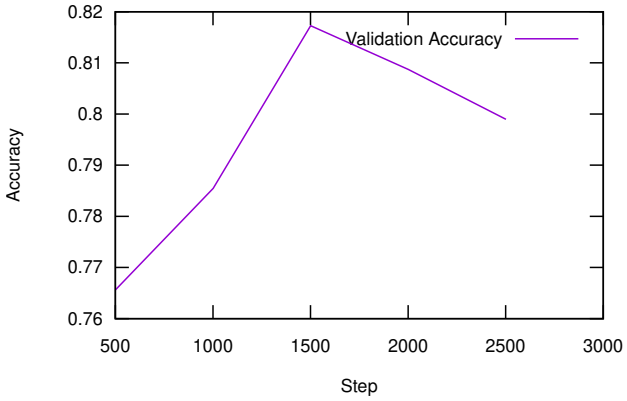

Fig. 7. Validation Accuracy for LeCun's model

5 ANALYSIS

The best model seems not to be the model proposed in LeCun, Huang, and Bottou (2004), but the more simple convolutional model adapted from other problem (MNIST). The difference in performance between the model proposed and my implementantion shows how difficult is to reproduce the results claimed in many papers on machine learning, but specially on Deep Learning, where the implementantion details really matter.

It is iteresting how all testing accuracy scores are bigger than the validation accuracy scores. This shouldn't happen, but because it always happens, I may conjecture that the set used for validation was slightly more difficult to learn than the set used for testing, even thought both come from the same dataset. A reason why this could have happened, is because I didn't randomize the set given for testing, I just broke the set in two equal parts, one for validation and one for testing. There is, probably, some order in which the data was stored in the testing dataset, leaving the "easiest" datapoints (images) at last.

6 CONCLUSIONS

A simple convolutional model seems to work better than an elaborated model, but this could change if a different algorithm for training is used or if the models are more fine tuned.

Any problem using images is a hard problem, and an annoying one, not only because the problem is annoying but because using manipulating images is expensive and prone to error.

It is much more stressful to work with Neural Networks than with other methods/techniques from Machine Learning, e.g., SVMs. It is very hard to know for sure how well a model works on a problem, or how to modify a model for a similar problem into the current one. Not knowing which direction to take or what went wrong when using NNs makes it very difficult to think forward.

A lesson I learned was to randomize, always randomize, the datasets that we are given, both training datasets and testing datasets. It is possible that in the order of the elements of the dataset some information is hidden, information about how the data was collected or stored. This is due mainly to prevent any weirdness in differences between validation and testing accuracy scores, like the ones present in this report.

Another lesson I learned in this assignment was that: reproducing results from papers in machine learning is hard!

REFERENCES

Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." *CoRR* abs/1704.04861. http://arxiv.org/abs/1704.04861.

LeCun, Yann, Fu Jie Huang, and Léon Bottou. 2004. "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting." In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), with Cd-Rom, 27 June - 2 July 2004, Washington, Dc, USA*, 97–104. doi:10.1109/CVPR.2004.144.