Project 1 of Deep Learning

Classification, weight sharing, auxiliary losses

Francis **Damachi** - Costanza **Volpini**

***Abstract*—The aim of this project was to show the impact of weight sharing and the use of auxiliary loss. The task was to compare two digits in a two-channel image. Starting from linear model until some more complex non-linear model, we have found out that convolutional neural networks represents the best solution to classify images.**

I. Introduction

Convolutional neural network (CNN) represents the most powerful model to analyse images. Indeed, CNN uses the pixel locality to learn better.

We have started the project training simple linear models, in particular *Linear Regression* and *Logistic Regression* (see Section IV) but we have notice that these models are too simple and not consider the pixel locality. We have then used a more complex model *Neural Network* with one-loss and two-losses (as explained and showed in Section V), we got a bad accuracy caused by the fact that we have a model composed by just linear layer. The last model that we have implemented is the *Convolutional neural network*, with this model we got a very good approximation (see Section VI). In all the models, we have used *Adam's method* instead of *stochastic gradient descent* as optimizer since it use an adaptive learning rate (the learning rate of SGD has an equivalent type of effect for all the weights/parameters of the model [1]). As criterion, we have used *Mean Square Error* for all the models.

For all the models we have explicitely separated in the code the feature extractor, used to process the input and extract high level features, from classifier(s), a simple linear layer that we have used to classify.

Section II explain the structure of the dataset, Section III describes briefly the content of each file. In Section VII, VIII we have showed the performance accuracy obtained by each model with a fixed seed, explaining why we got these results.

II. Data

The data set consists of an input series of $2 \times 14 \times 14$ tensor (2 images in grayscale). Training and test set is 1000 pairs each. In total we will have six tensors: images ($N \times 2 \times 14 \times 14$), prediction (boolean), classes of the two digits ($N \times 2$), both for training and test sets. The MNIST database is composed by handwritten digits (from 0 to 9); to load the dataset we have used the provided function `generate_pair_sets(N)`.

III. Code structure

In this section contains the description of each file.

- `scripts/data.py`: class to generate the dataset. Contains different methods (e.g. flat the input, get a dataset in 2D or in 3D, enable the hot-encoding).
- `scripts/model.py`: general class to define a model to train and test it (with corresponding plot and history).
- `scripts/models_implemented.py`: contains all the model classes (e.g. NNModel1Loss, CNNModel2Loss).
- `main.py`: contains examples of each models, in order to call and train a model.

Files `scripts/cross_validation_report.py` and `BoxPlot_generator.ipynb` are made for report purpose.

IV. Linear Models

We have implemented *Linear Regression* and *Logistic Regression*. Regression is represented by a final layer that it is linear. In the logistic regression we have used a sigmoid as an activation function in the output layer. A linear layer accept as input a vector of values, that is the reason why we need to flat the input (the raw input is composed by 2D images). In fact a linear layer give a weight to each pixel, then it will accept a vector composed by N rows and D dimensions (columns), in our case it will have size $14 \times 14 \times 2$. Linear layers bad generalize images, since it just memorize the train images, that causes a good train accuracy but a very low test accuracy.

During the train of linear models we have noticed that sometimes the accuracy and loss could both decreases, that can happen if we have a case like the blue line in Fig. 1. Linear Regression has a strange behavior due to the fact that it has not have a sigmoid and the loss MSE works very bad in a classification task without mapping the value in a range.

V. Neural Network Models

We have implemented two models for the neural network, the first one with 1 loss and the second one with 2 losses, as showed in Fig. **??** and Fig. **??**. Both models use the same feature extractors but they differs for the classifier. The first one has a boolean classifier that returns 1 if the first image is smaller then the second one. The second model, it has also a digit classifier that returns the corresponding digit for both the handwritten images (we have used an hot encoding to give a target to each node). For these two models the images inputs are passed as 2D images. Since the feature extractor processes both images together (it does not consider the two images independently), we can use a digit classifier with 20 nodes as output (10 nodes for each images; i.e. from 0 to 9 as value of

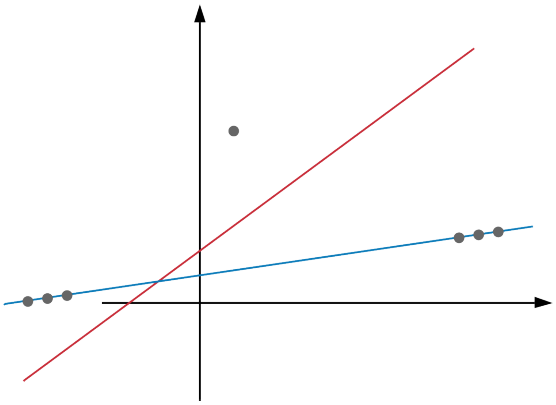

Fig. 1: Red line has an higher loss but a better accuracy. Instead, blue line has a lower loss but it predicts bad (lower accuracy). We can conclude saying that blue line fits well samples that already known.

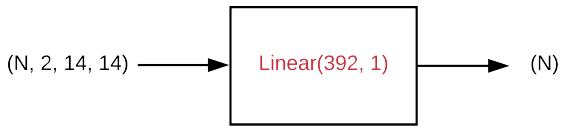

Fig. 2: Linear Regression model

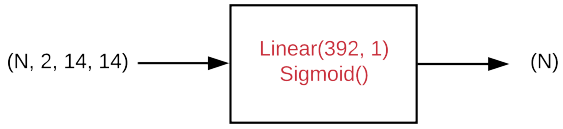

Fig. 3: Logistic Regression model

digit). We have noticed that NN with 2 losses seems to perform better during the cross validation but not in the test (see Table II). That seems to be in contrast with what we would expected; the cause could be found in the structure of the model, since we just have linear layers we could have improved the computation considering both the images independently. We have then decided to try with a convolutional neural network, since *conv3d* allow to automatically process both the images independently. We have observed that two losses could improve the accuracy if the model is more complex, otherwise it would affect negatively the accuracy (as in our case).

VI. Convolutional Neural Network Models

Two models of convolutional neural networks were trained, both received as input 3D images since we have decided to use a *conv3d*. Conv3d are powerful cnn filter that are able to traverse the image in x, y and z axes; that make possible to process both the two images in a independent way. CNN are powerfull since they use pixel locality, then instead of flatting the input as done in previous model we can just consider the 3D raw images. As done, in Sec. V we have decided to compare two models, one with one loss and the other one with two losses, as done with Neural Network models. We have used the dropout to generalize better and then obtain a more robust model. CNN with 2 losses seems to return the best accuracy (see Table II).

TABLE I: Table with number of parameters for each model.

		Parameters
Models	Linear Regression	393
	Logistic Regression	393
	NN (1 loss)	133633
	NN (2 losses)	136213
	CNN (1 loss)	47803
	CNN (2 losses)	49773

TABLE II: Table accuracy (Cross validation: mean $\pm$ std).

		Accuracy	
		Cross validation	Test
Models	Linear Regression	61.8 ± 7.467	69.2
	Logistic Regression	72.4 ± 0.860	73.8
	NN (1 loss)	70.5 ± 12.696	80.5
	NN (2 losses)	73.8 ± 10.419	78.4
	CNN (1 loss)	81.6 ± 2.289	81.7
	CNN (2 losses)	81.6 ± 2.223	83.4

VII. Comparison of models

Since linear models are really sensible to the weights setting we have decided to use a cross validation to compare the models (however, we did not implement our cross validation with *pytorch*, we have used the library *sklearn*). Table I summarizes all the number of parameters for each model. In the plotboxs (see Fig. 4), it is possible to see the accuracies for each models (using cross validation). We can notice that the accuracy vary a lot in simple models since they highly depend from weights initialization.

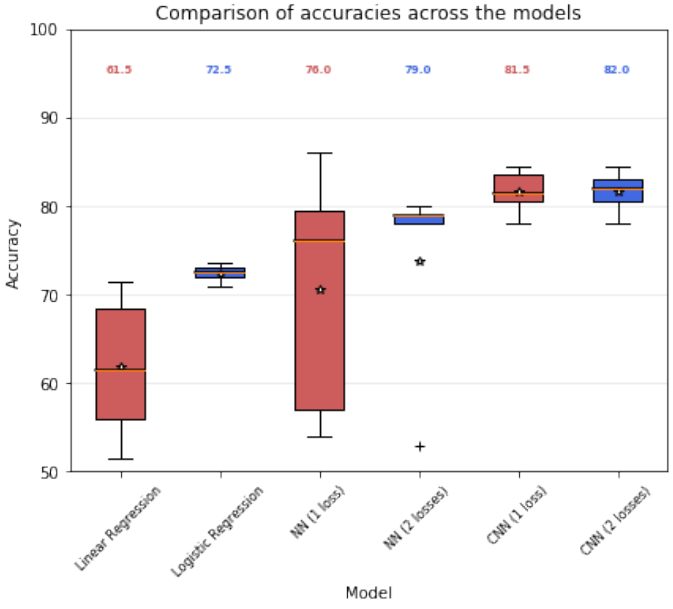

Fig. 4: Plotbox that show the accuracy using cross validation. Star symbols represent the average values.

Table II shows the comparison between the accuracy obtained during the cross validation and the one of test. We can notice that test accuracy got better scores because we have used a bigger dataset to train. The dataset used in cross validation was $4/5$ of the total train set, instead the one used to test at the end was $5/5$ (all) of the dataset.

VIII. Conclusion

Then CNN represents the best model with images recognition. In particular, the CNN with 2 losses seems to perform slightly better. As future improvement we would like to implement the cross validation using the *pytorch* library.

References

[1] optim.Adam vs optim.SGD. Lets dive in. https://medium.com/@Biboswan98/optim-adam-vs-optim-sgd-lets-dive-in-8dbf1890fbdc