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 with around 15% error rate.**

I. Introduction

Convolutional neural network (CNN) represents the most powerful model to analyse images because it preserves the dimension of an image, that guarantees a good use of pixel locality to learn better. Simple linear models were used at the beginning of the project, in particular *Linear Regression* and *Logistic Regression* (see Section IV), we obtained an accuracy of 70% (these models 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 better accuracy (80%), but we were just using linear layer. Our goal, was to try to take advantage by the locality of the pixel (we want to analyze the context of an image and not just a single value). The last model, takes advantage by the context of an image, *Convolutional neural network* we got a very good accuracies $(82 - 84\%)$ (see Section VI).

In all models, we have used *Adam's method* instead of *stochastic gradient descent* as optimizer since it uses 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*.

The models implementation consists of an abstract class that acts as super class. Then, for each model we have implemented a class and inside them we have explicitly separated 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 explains the structure of the dataset, Section III describes briefly the content of each file. Section VII explain which auxiliary loss we have used. In Section VIII and IX we have showed the performance accuracy obtained by each model using a cross validation, 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

This section describes the content of each file.

- `code/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).
- `code/model.py`: general class to define a model to train and test it (with corresponding plot and history).
- `code/models_implemented.py`: contains all the model classes (e.g. NNModel1Loss, CNNModel2Loss).
- `main.py`: contains examples of each model, in order to call and train it.

Files `code/cross_validation_report.py` and `BoxPlot_generator.ipynb` are made for report purposes.

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 accepts 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). It gives 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 $2 \times 14 \times 14$. Linear layers bad generalize images, since it just memorizes the train images, that causes a good train accuracy but a very low test accuracy.

Linear Regression has a strange behavior in training 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.

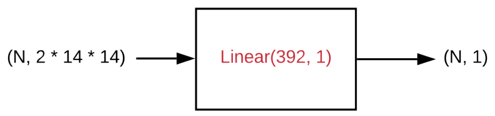

Fig. 1: Linear Regression model

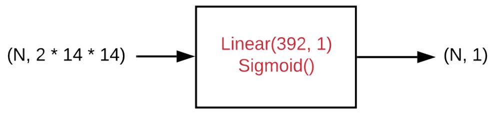

Fig. 2: Logistic Regression model

V. Neural Network Models

We have implemented two models for neural network, the first one with 1 loss and the second one with 2 losses, as showed in Fig. 3 and Fig. 4. Both models use the same feature extractors but they differs for the classifier(s). The first one has a boolean classifier that compares the two digits visible in the two images. The second model, it has also a digit classifier that returns the corresponding digit for both the handwritten images (class classification: we have used an hot encoding to give a target to each node). For these two models the 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 digit). We have noticed that NN with 2 losses seems to perform worst during the test then the model with 1 loss (see Table II). That seems to be in contrast with what we would expected; two losses could improve the accuracy if the model is more complex, otherwise it would affect negatively the accuracy (as in our case). Since we just have linear layers we could have improved the computation considering both the images independently. We have then decided to implement a convolutional neural network, since *conv3d* allow to automatically process both the images independently.

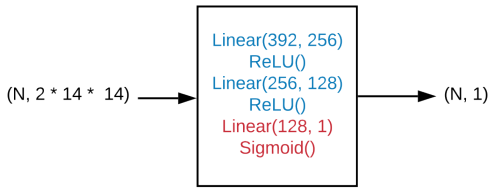

Fig. 3: Neural Network model (1 loss)

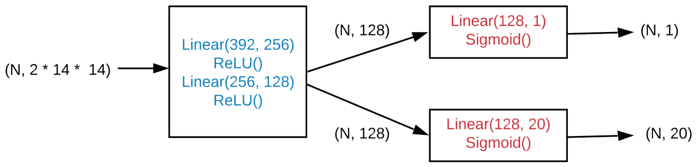

Fig. 4: Neural Network model (2 losses)

VI. Convolutional Neural Network Models

Two models of convolutional neural networks were trained, both received an input of 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 independently process both the two images. CNN are powerful since they use pixel locality, instead of flatting the input as done in previous model we consider the two 3D images. As done with the model of NN, in Sec. V, we have decided to compare two models one with one loss and the other one with two losses (see Fig. 5 and Fig. 6). We have used the dropout to generalize better and then obtain a more robust model. CNN with 2 losses got the best accuracy with 84% (see Table II).

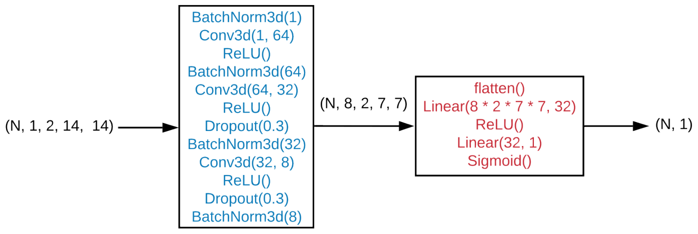

Fig. 5: Convolutional Neural Network model (1 loss)

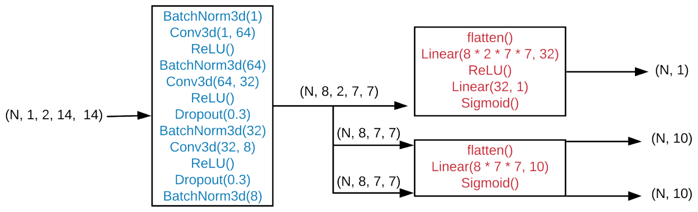

Fig. 6: Convolutional Neural Network model (2 losses)

VII. Comparison between 1 loss and 2 losses

We have decided to train the Neural Network and Convolutional Neural Network models also with an auxiliary loss. In addition of the loss obtained with the boolean classifier, we have added a new classifier that should detect the digit class of both images (the target is then an n-hot-encoding). Solving two tasks in parallel can help the model to extract a better representation of the data. The use of an auxiliary loss improved of 2% the accuracy in CNN model.

VIII. 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. As expected neural networks require more parameters to got a good accuracy since they do not have weight sharing. Convolutional neural networks use weight sharing (a weight for all the pixels of the same filter) then it requires less parameters to got a good accuracy.

Plotboxs (see Fig. 7) show 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.

Table II shows the comparison between the accuracy obtained during the cross validation and the one in test. We can notice that test accuracy got better scores because we have used a bigger dataset to train. The dataset used in cross

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)	51733

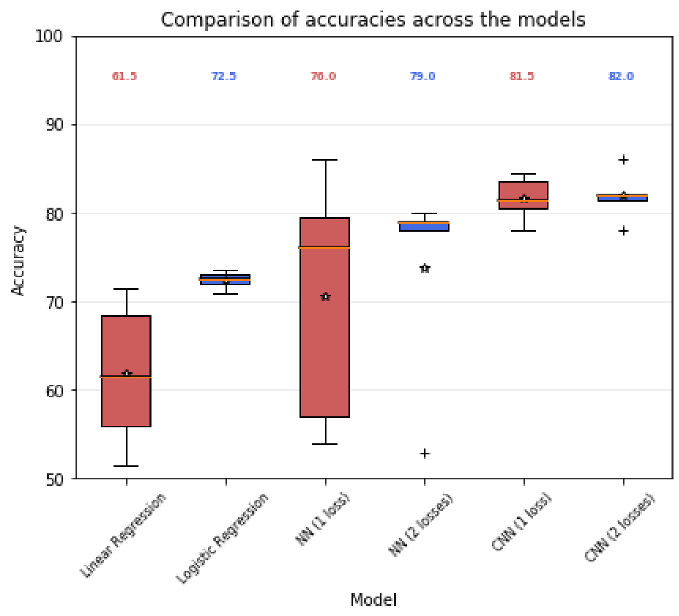

Fig. 7: Plotbox that show the accuracy using cross validation. Star symbols represent the average values.

TABLE II: Table accuracy (Cross validation: mean ± 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)	81.9 ± 2.538	83.7

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.

IX. Conclusion

Convolutional Neural Network represents the best model with images recognition. In particular, the CNN with 2 losses seems to perform slightly better since it is able to extract better the data. We have seen that weight sharing (CNN) improves the accuracy and the robustness of the model, the use of an auxiliary loss in this context improves the results obtained.

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