Deep Learning Project 1
Classification, weight sharing, auxiliary losses

Célia Benquet, Sven Borden & Artur Jesslen

***Abstract*—The deep learning field has seen tremendous amount of new architectures of neural networks (NNs) and optimization techniques in the last years. This project aims at recognizing digits from the MNIST dataset and compare them by experimenting some of those architectures. We implemented 4 NNs, a linear NN and 3 convolutional NNs (LeNet, ResNet, AlexNet). We also implemented 2 regularization techniques, dropout and batch normalization as well as an auxiliary network on LeNet. The results show that the ResNet network using batch normalization with 2 residual blocks gives the best predictions.**

I. Introduction

Over the last few years, the complexity of the tasks achieved by the NNs has become higher and higher. For a long time, the solution was to build deeper neural networks. Recently, new architectures and new techniques have been developed to improve performances and to be able to achieve those tasks. This project aims at comparing some of those architectures and techniques on the MNIST dataset. The networks are firstly trained to recognize the digits and the comparison between two digits can be done using basic arithmetic. As the dataset is simple, experiments can be ran rapidly even with training and testing sets of 1,000 pairs each and heavy nets are not necessary to obtain good predictions. Hence, we focus on small and easily trainable networks, designed for our dataset directly, as it is computationally easier to implement different techniques as well as comparing a wide range of hyperparameters.

We choose to implement four neural networks, with different variant for each. First, a simple linear neural network is used to show the impact of weight sharing. By comparing its results to those of the other NNs, all being convolutional neural networks (cNNs). Then LeNet (Lecun et al., 1998 [1]) and ResNet (Kaiming et al., 2015[2]) are implemented by trying out different techniques such as batch normalization (BN), dropout and connection skipping using a residual net in order to optimize the output performance. Finally, we adapt the AlexNet (Krizhevsky et al., 2012 [3]), designed for the ImageNet dataset, of input size $224 \times 224 \times 3$, to be performed on our dataset of size $14 \times 14 \times 1$. As it is already heavier in its number of parameters, and so in the computational power needed, we choose not to modulate the parameters and different techniques.

II. Methods

A. Presentation of the data

The data consists in a set of 14×14 grayscale images from the MNIST dataset. The training set has a size of 1,000 pairs of images.

B. Deep-Learning methods

The project was done using PyTorch, an open source machine learning library based on the Torch library [4]. The goal is to implement a deep network such that, given a pair of grayscale images, representing digits, we are able to predict if the first digit is lesser or equal to the second. We decide to divide the problem in two sub-problems. First, we design a deep network, trained on MNIST to recognize digits based on their groundtruth. Then, using this classification, we can easily compare 1,000 pairs of those digits.

We test 4 different architectures of deep networks: a linear network, LeNet, ResNet and AlexNet, for which we try different parameters and implement batch normalization, dropout and an auxiliary loss. We tried implementing a Xception network but we met computational power limitations. The different combinations that we tried are summarized in Table I. We first ran each set-up for a dataset of 350 inputs. Then, we selected the optimal combinations and re-ran the simulation for 1000 data points. For all the NNs, optimization is performed using a cross-entropy loss and Stochastic Gradient Descent (SGD) (more precisely an SGD optimizer was used for each network except for the linear one).

1) Linear Neural Network: Our linear architecture has a scalable number of layers, all fully-connected, with the same number of nodes (also scalable). We implemented the dropout regularization technique (Srivastava et al., 2014 [5]). Usually, in complex NNs, units evolve so that they compensate one another's mistakes. The notion of dropout was introduced to compensate overfitting by preventing co-adaptation and making the presence of other hidden units unreliable. Even if cNNs are more robust for image classification than linear layers, the MNIST dataset is very simple and consistent and so we expect linear NNs to perform good on it (42018 parameters).

2) LeNet Neural Network: LeNet has 2 convolutional layers combined to pooling layers as well as 3 fully-connected layers (about 60,000 parameters). Even if the original paper from Lecun et al. (1998 [1]) makes use of a *Tanh* activation function, we use the *Rectified Linear Units (ReLU)* due to its lower computational cost and to avoid vanishing gradient problems (Glorot et al., 2011 [6]). We implement the dropout and batch normalization (Ioffe et al., 2015 [7]) regularization techniques. This last one also increases training efficiency. It consists of explicitly forcing the activation statistics during the forward pass by re-normalizing them so that the following layers do not need to adapt to their drift. We experimented all possible combinations as recommended by Garbin et al. (2020 [8]).

We also implemented an auxilliary loss between the second and third convolutional layers. Considering a network built by stacking up identical modules, one can attach small networks to the output of each modules (Szegedy et al., 2014 [9]). Those auxilliary network's task is to predict the same label as the final module. It promotes learning for each module and increases training efficiency by avoiding gradient vanishing. Finally, we modulate the kernel size to optimize the output.

3) ResNet Neural Network: Due to the effect of vanishing/exploding gradient, when facing deeper NNs, training accuracy gets saturated and degraded rapidly. This effect is called the *degradation* problem. To overcome the effect, residual cNNs are designed. "Shortcut connections", integrated to a cNN, skip layers without adding extra parameters nor computational complexity (He et al., 2015 [10]). We implement a residual cNN architecture for which we modulate the structure. It is composed of one convolutional layer, one fully-connected layer as well as residual blocks. The number of channels of the convolutional layer, the kernel size of the convolutions and the number of residual blocks are optimized. Each residual block consists in two convolutional layers. Dropout and batch normalization are implemented and their combinations evaluated. Despite the complexity of this network, it is composed of only about 50,000 parameters.

4) AlexNet Neural Network: The original AlexNet NN has around 60M parameters, which makes it a larger, computationally heavier NN than the previous ones. AlexNet has 8 layers — 5 convolutional and 3 fully-connected - and was originally designed to classify the Imagenet dataset, whose images have a larger size (Krizhevsky et al., 2012 [3]). Hence, we adapt the network to be able to take smaller images from the MNIST dataset as input. We obtain a NN of about 1,8M parameters.

III. Results

The best combinations for the 4 NNs are in bold in Table I. The results that correspond to best epochs for those combinations are presented in Table II. Our results are obtained using the following parameters:

- 20 repetitions.
- 25 epochs
- 1000 pairs of images for both train and test sets.

We also present some interesting comparisons of implementations that are discussed in the section IV. Table III compares 2 linear NNs composed of 2 linear layers of 128 nodes each (best combination from Table I) with and without dropout. The predictions are similar. Table IV compares 2 LeNet networks composed of convolutional layers with a kernel size of 5, batch normalization with and without an auxiliary loss. The predictions are slightly better with the auxiliary loss.

NN	Train	Test (digits)	Test (comparison)
Linear	$0.022 \pm 0.101\%$	$6.802 \pm 0.608\%$	$4.590 \pm 0.551\%$
LeNet	$0.160 \pm 0.097\%$	$2.962 \pm 0.431\%$	$2.330 \pm 0.548\%$
ResNet	$0.238 \pm 0.160\%$	$2.803 \pm 0.402\%$	$2.085 \pm 0.431\%$
AlexNet	$1.240 \pm 0.760\%$	$4.160 \pm 0.620\%$	$3.150 \pm 0.310\%$

Table II: Results of best epochs of best combination for each NN architecture. The results are composed by their means and standard deviation.

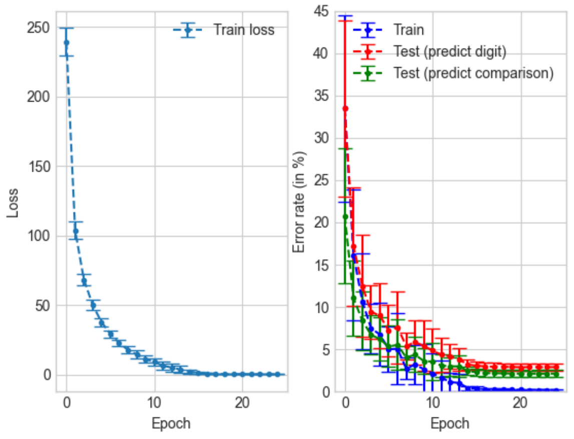

Figure 1: Mean and standard variation of the training loss, training error and test error (digit prediction and comparison prediction) are represented during the training (25 epochs, 20 repetitions, a dataset of 1000 pairs (train set and test set)) for the ResNet Neural Network.

NN	Train	Test (digits)	Test (comparison)
Linear without Dropout	$0.022 \pm 0.101\%$	$6.802 \pm 0.608\%$	$4.590 \pm 0.551\%$
Linear with Dropout	$0.075 \pm 0.094\%$	$6.392 \pm 0.765\%$	$4.595 \pm 0.772\%$

Table III: Comparison of linear NNs of 2 linear layers and 128 nodes per layer (best combination) with and without dropout. The predictions are similar but we keep the linear NN without dropout as the best combination because it gives a lower error rate on the comparison. The results are composed by their means and standard deviation.

NN	Train	Test (digits)	Test (comparison)
LeNet without Aux. Loss	$0.243 \pm 0.117\%$	$3.283 \pm 0.452\%$	$2.390 \pm 0.594\%$
LeNet with Aux. Loss	$0.160 \pm 0.097\%$	$2.962 \pm 0.431\%$	$2.330 \pm 0.548\%$

Table IV: Comparison of 2 LeNet NNs composed of convolutional layers with a kernel size of 5 and batch normalization with and without an auxiliary loss. The predictions are better with the auxiliary loss. The results are composed by their means and standard deviation.

IV. Discussion

Before going deeper inside the discussion, we note that the random initialization may change the results by a few tenth of percent from one trial to the other. This small variation could change the order of the best models as well as the best combination for each architecture. Similar results have to be analyzed carefully.

A. Architecture comparison

Linear NNs' results are not as good as cNNs. This can be explained by the fact that linear NNs do not have convolution and in this way, the notion of neighborhood does not exist. On the contrary, cNNs can recognize an object even after a translation.

As AlexNet architecture has more parameters than LeNet and ResNet, the training time before convergence is expected to be higher and it would explain the poorer results using this architecture.

We observe that ResNet performs slightly better than LeNet but the difference might be more visible for higher complexity tasks, such as classification of images from the ImageNet dataset.

Neural Net	BN	Dropout	Residual block	# Channels	Kernel size	# Nodes	# Linear layers	Auxiliary Loss
Linear	-	True **False**	-	-	-	[16, 32, 64, **128**, 264]	[1, **2**, 3 5]	-
LeNet	**True** False	True **False**	-	-	[3, **5**]	-	-	**True** False
ResNet	**True** False	True **False**	[1, **2**, 3, 4, 5]	[1, 2, 4, 8, **16**]	[3, 5, **7**]	-	-	-
AlexNet	-	-	-	-	-	-	-	-

Table I: Performed experiments. All combinations were experimented for an initial data size of 350. The results were analysed and the best for each NN was re-simulated for a bigger data set of 1,000. The best combination for each neural net is represented in bold.

B. Impact of weight sharing

A standard linear network has the advantage to be general and powerful but it has a lot of parameters and consequently needs a lot of data to be trained. Weight sharing allows a reduction of the number of parameters as well as a more robust feature detection by "locally" processing the data in a sparser manner. For that, we use convolutional layers. One can see that the 3 cNNs we implemented give better results than linear NNs. Indeed, the MNIST dataset consists in digits with different calligraphy and these differences of the digits need to be taken into account to have a robust classification.

By using dropout, one creates multiple implicit ensembles that share weights by randomly removing a given number of units in the layer. We could expect from dropout on a linear NN to have a similar effect than weight sharing, by obtaining better predictions. However, the results are overall similar (see Table III) and we keep the linear NN without dropout as the best combination because it gives a lower error rate on the comparison. Indeed, dropout aims at compensating overfitting, which was not the case for our linear NN on images as the train error was not increasing.

C. Effect of auxiliary loss on training

As expected, we have better predictions using an auxiliary loss on LeNet (see Table IV). However, the effect of the auxiliary loss might lead to real improvements in the results with a deeper network, where the vanishing gradient has a higher effect than for a NN as LeNet. The method was originally implemented for the GoogleLeNet, a deeper network. Computational limitations prevent us to assess more about auxiliary losses performance improvements.

D. Dropout versus Batch Normalization

According to Garbin et al. [8], dropout and BN have to be used in cNNs by first experimenting the different sets of combinations. They explain that, when in doubt and short on time to experiment, it is safer to only use BN. Here, after testing the different combinations, we found BN without dropout to be the best. Once again, in order to appreciate the effects of dropout, the NN would have had to be deeper and the number of epochs higher in order to observe more important effects. As well as having a regularization effect, BN also allows a higher flexibility in the weights initialization and choice of the learning rate. Therefore, it should not be seen as a way to improve performances, but more as a way to train your model faster (in terms of convergence speed) thanks to its regularization effect. Hence, it could make sense that the best combination is BN without dropout.

References

[1] Yann Lecun. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[2] He Kaiming. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[3] Alex Krizhevsky. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 2012.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. *PMLR*, pages 315–323, Jun 2011.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Cornell University*, 2015.

[8] Christian Garbin, Xingquan Zhu, and Oge Marques. Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia Tools and Applications*, 2020.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv*, Sep 2014.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv*, Dec 2015.