EPFL

Deep Learning EE-559

Project 1: Classification, weight sharing and auxiliary loss

Clémentine Lea Aguet, Yamin Sepehri, Mahdi Nobar

May 16, 2019

Abstract

The ultimate goal of this project is to evaluate the performance of different learning architectures. The models built should be able to infer from a given pair of gray-scale images which digit value is bigger. In order to achieve this goal, various approaches are implemented incorporating different architectural components. The simplest model is a fully connected network. Convolution, multi-layer and auxiliary losses are the following concepts integrated. After comparing the different performances, a multi-layer convolution network with auxiliary losses appeared to be most efficient for this particular task.

Keywords: Deep Learning, Network architecture, Weight sharing, Auxiliary loss, *PyTorch*

1 Introduction

Deep learning has become very popular and successful in various research domains or industrial applications. Achieving great performance particularly depends on the design of the neural network architecture. Furthermore, the design of the model is also critically dependent on the application. The main objective of this project is to understand the impact of different network architectures on a simple binary classification task, which consists in comparing pairs of digits. For this purpose, multiple neural networks are built using *PyTorch* framework and their performances analyzed. The main idea of our approach is to proceed in a progressive and iterative manner. Starting from a simple *fully connected* network, the architecture of the network then becomes more and more complex in order to improve the performance of the model.

One can first try to change the type of layer, using convolution instead of fully connected. Inputs like images typically contain some invariance in translation, meaning that a representation important at a certain spatial location should also be useful at other locations. Convolution layer was developed based on this assumption. It applies some linear transformations locally and across the input, while preserving its structure. Such type of layer thus benefits from *weight sharing*, which consists in using fixed weight for the kernel sliding across the input. It significantly reduces the number of parameters to optimize during the training phase. Thus allowing to converge faster and to learn deeper architectures more easily.

The next approach is to implement a deeper architecture, adding more layers to this convolution neural network (CNN). Examples have already demonstrated that the learning process can considerably benefit from deeper networks. Adding layers can allow to extract more abstract and complex features useful for the learning. It is nevertheless important to remember that the network architecture and its chosen depth is highly conditioned by the desired task to perform.

The final idea investigated in this project is to exploit auxiliary loss. This approach is more human-like reasoning oriented but its feasibility depends on the data available. In human reasoning, additional conditioning or past knowledge greatly help to achieve a specific task [1]. One possibility to improve the performance of a model could thus be to incorporate auxiliary information into the learning algorithm.

2 Dataset

The dataset available for this project contains pairs of gray-scale images coming from the *MNIST* handwritten digit database. Input tensors are of size $N \times 2 \times 14 \times 14$. Their label tensors have a size N with content in $\{0, 1\}$, indicating whether the first or the second digit is larger. Additional information is available, it is a classes tensor of size $N \times 2$ with values in $\{0, ..., 9\}$, revealing the two digits present in the pair.

The training and test sets are 1000 pairs each, with their corresponding labels and classes. In order to perform some hyperparameter tuning and to assess a more global idea of the model performance, the training sets is randomly separated into 800 samples for training and 200 samples for validation. The test set itself is used to evaluate the performance of the model on a completely new dataset, unseen during the training phase.

3 Methods

This section describes the different approaches and network architectures implemented in this project. No matter the architecture, all models are trained with an *Adam* optimizer, a *CrossEntropy* loss function and processed by batches.

Adam optimizer or adaptive moment estimation [2] is an adaptation of Stochastic gradient descent. This approach calculates different learning rate for each parameter based on approximations of the first and second moments of the gradients.

CrossEntropy loss function defines the performance of a classification with outputs being probability between $[0, 1]$. With classical *MSE*, miss-classification would not be punish sufficiently. *CrossEntropy* is thus more appropriate for binary classification task.

All models also share some components. Typically the activation functions. *ReLU* is used as activation function in most hidden layers. It has the property to be non-saturating with no vanishing derivative and sparse coding. *Sigmoid* activation function is considered in the prediction layer. Outputting in the range $[0, 1]$, this function is well suited for binary classification problem, as it is the case here.

Dropout is commonly added as a deep regularization technique. It proceeds by removing units with a probability p during the forward pass on each sample. It thus reduces the dependencies between hidden units, which is a source of over-fitting [3]. Dropout is not applied during model evaluation and all the units are put back.

3.1 Fully Connected model (FC)

As a first step, a simple fully connected network is implemented, see architecture in Table 1 and in Appendix 2. In such type of layer, neurons of adjacent layers share all connections, but neurons within a single layer are not connected. The pairs of *MNIST* images with dimension $2 \times 14 \times 14$ (2 channels for the 2 digits to compare) are first flatten before been given as input to the model.

Layer	Type	Parameter	Output size
	Input		N x 2 x 14 x 14
1	Flatten		N x 392
	Linear		N x 32
	Dropout	p = 0.4	
	ReLU		
2	Linear		N x 2
	Sigmoid		

Table 1: *Fully Connected network*

3.2 Convolutional model (CNN)

The second approach is to implement a convolutional neural network (CNN), see architecture in Table 2 and Appendix 3. The network takes similar inputs as the fully connected one, however flattening is not required at that stage. A CNN typically involves convolution, pooling and fully connected layers.

Pooling layer is useful for low dimensional computation starting from high-dimensional input. It will group multiple activations into a more meaningful one, typically using a max operation. This technique provide pseudo-invariance to local deformations.

Layer	Type	Parameter	Output size
	Input		N x 2 x 14 x 14
1	Conv2D	k = 5, p = 1	N x 32 x 12 x 12
	ReLU		
	MaxPool2D	k = 2	N x 32 x 6 x 6
	Dropout	p = 0.4	
2	Flatten		N x 1152
	Linear		N x 64
	ReLU		
	Dropout	p = 0.4	
3	Linear		N x 2
	Sigmoid		

Table 2: *Convolution network*

3.3 Deep Convolutional model

The next architectural change consists of building a deeper network, as adding layers often help to improve the performance. A second convolution layer is thus joint to the preceding convolution model. Its architecture can be visualized in Table 3 and Appendix 4. It is inspired by *LeNet-5* [4] architecture, which is a good network when training on *MNIST* database.

Layer	Type	Parameter	Output size
	Input		N x 2 x 14 x 14
1	Conv2D	k = 5, p = 1	N x 32 x 12 x 12
	BatchNorm2D		
	ReLU		
	MaxPool2D	k = 2	N x 32 x 6 x 6
	Dropout	p = 0.4	
2	Conv2D	k = 5, p = 1	N x 64 x 4 x 4
	BatchNorm2D		
	ReLU		
	MaxPool2D	k = 2	N x 64 x 2 x 2
	Dropout	p = 0.4	
3	Flatten		N x 256
	Linear		N x 64
	ReLU		
	Dropout	p = 0.4	
4	Linear		N x 2
	Sigmoid		

Table 3: *Deep Convolution network*

While increasing the number of layers, the risk of gradient vanishing appears. The weights are not updated as the gradient remains really close to 0. Choosing a small learning rate and using some *ReLU* activation function can help. Incorporating *Batch Normalization* is another option to address this issue. This technique aims to maintain proper statistics during forward pass using normalization. It proceeds by shifting and scaling, both determined by the mean and variance computed on the batch. The position of such layer in the network relative to non-linearity is still an ongoing discussion.

3.4 Auxiliary model

As described in the introduction, the last idea consists in integrating auxiliary information to the learning process. The details of the architecture is described in Table 4 and Appendix 5.

Additionally to the expected labels indicating the largest digit within each pair, the classes defining both numbers of the pair are used. While training the model, two tasks are evaluated. The first one is how

the network is capable to extract digits from images. The second and final task refers to the classification of the largest digit of the pair. The network architecture starts with a part similar to the deep convolution network previously implemented at the exception of its input and the final layer. It takes as inputs single images. And the prediction layer outputs a $[N \times 10]$ tensor (one value of each possible digit) passed through a *Softmax* activation function. Such function is more suitable than *Sigmoid* activation function in multi-classification problem. In the second part of the network, the first outputs computed for each image of the pair are concatenated to create a $[N \times 20]$ tensor. A fully connected layer followed by a *Sigmoid* activation function outputs the final prediction.

Layer	Type	Parameter	Output size
	Input		[N x 1 x 14 x 14] x 2
1	Conv2D	k = 5, p = 1	[N x 32 x 12 x 12] x 2
	BatchNorm2D		
	ReLU		
	MaxPool2D	k = 2	[N x 32 x 6 x 6] x 2
	Dropout	p = 0.4	
2	Conv2D	k = 5, p = 1	[N x 64 x 4 x 4] x 2
	BatchNorm2D		
	ReLU		
	MaxPool2D	k = 2	[N x 64 x 2 x 2] x 2
	Dropout	p = 0.4	
3	Flatten		[N x 256] x 2
	Linear		[N x 64] x 2
	ReLU		
	Dropout	p = 0.4	
4	Linear		[N x 10] x 2
	Softmax		
5	Concatenate		N x 20
	Linear		N x 2
	Sigmoid		

Table 4: *Auxiliary network*

The auxiliary loss compares the outputs of the first network section for the two images to their respective classes (L_{Aux_1} and L_{Aux_2} in Equation 1). While the second loss compares the final outputs to the labels (L in Equation 1). A combination of the losses is computed following Equation 1 and used in the backpropagation pass to update the parameters of the model. As the main goal of this project is to perform a digits comparison, the loss L has a higher contribution to the global loss.

$$L_{global} = 0.2L_{Aux_1} + 0.2L_{Aux_2} + 0.6L \qquad (1)$$

4 Results & Discussion

All different architectures discussed are trained on the dataset described above. In order to be able to compare their performances, most hyper-parameters are kept similar across networks. The values are defined after some parameter tuning, varying the learning rate between $[0.01, 0.0001]$, the batch size between $[10, 200]$ and the number epochs. The model accuracy as well as a plot of the loss vs the number of epochs, such as in Figure 1, help to select them. A learning rate of 0.005, a batch size of 100 and 100 or 200 epochs according to the model complexity were chosen. Models are run over 10 iterations and their averaged performances on the training and test sets are represented in Table 5.

	Train		Test	
	Mean	Std	Mean	Std
FC	0.9500	0.91	0.7401	1.41
FC + Dropout	0.9612	1.17	0.7642	1.12
CNN	0.9881	0.54	0.8251	0.91
Deep CNN	0.99	0.40	0.8301	1.36
Auxiliary model	0.9869	0.62	0.9558	1.11

Table 5: *Different models accuracy*

Comparing the results of the Fully connected network with or without *Dropout*, one can understand the impact of incorporating such layer in the architecture. While not significantly changing the training accuracy, it improves the test accuracy and thus reduces the overfitting.

Results in Table 5 also demonstrate that changing for convolution layer has a positive effect on the performance. The model takes advantage of the weight sharing concept, which also allows to lower the difference between training and test accuracy.

However adding more layers to the architecture does not seem to have a huge impact here. With a simple task, the model does not need to learn complex representations. A deeper network will not improve the performance but could increase the risk of overfitting.

The most accurate architecture implemented is the auxiliary model. Its performance is illustrated in Figure 1. The validation loss and accuracy remain close to the training ones during the learning. It proves the potential of using auxiliary information if any available.

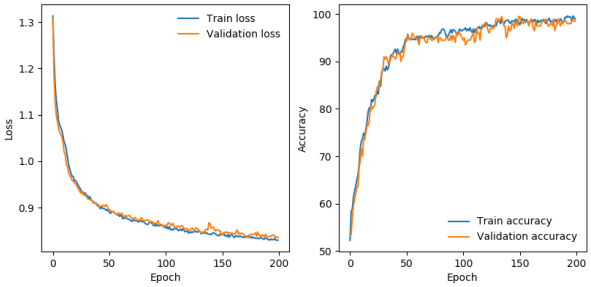

Figure 1: *Losses and accuracy of auxiliary model trained with batch size = 100 and learning rate = 0.005*

Siamese model is also an architecture investigated in this project. However the results obtained were not significantly relevant in comparison to those of the other models. More time and effort should be invested in this approach, particularly in the selection of the loss function and some parameter tuning to reach good performances. Nevertheless this structure seems well appropriate for this task.

5 Conclusion

Testing different architectures allows to understand the purpose and impact of diverse network components on a simple task performance. Even though adding auxiliary information gives the best performance, this model is still not completely optimized. Further parameter tuning with cross-validation should address other elements of the architecture, such as the number of hidden units, the kernel size, the loss or the number of layers.

References

[1] F. Wang and C. Song. A neural network with logical reasoning based on auxiliary inputs. *Frontiers in Robotics and AI*, 2018.

[2] D.P. Kingma et al. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2017.

[3] N. Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[4] Y. LeCun et al. Gradient-based learning applied to document recognition. *Proc of the IEEE*, 1998.

[5] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. MIT Press, 2016.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science, 2006.

Appendix

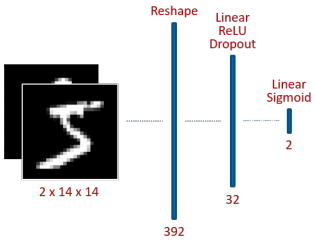

Figure 2: *Fully connected model architecture*

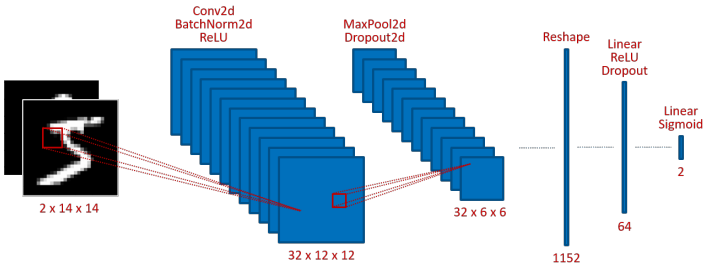

Figure 3: *Convolutional model architecture*

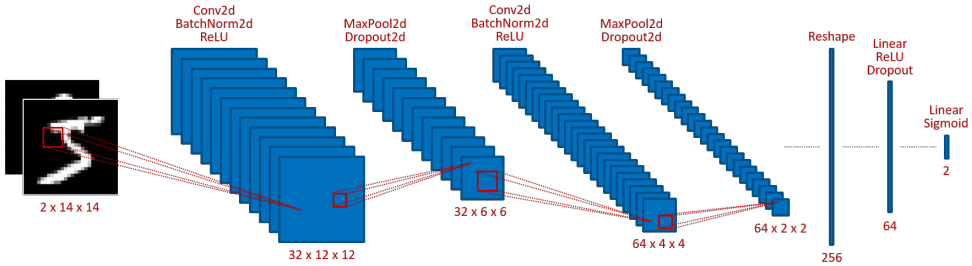

Figure 4: *Deep convolutional model architecture*

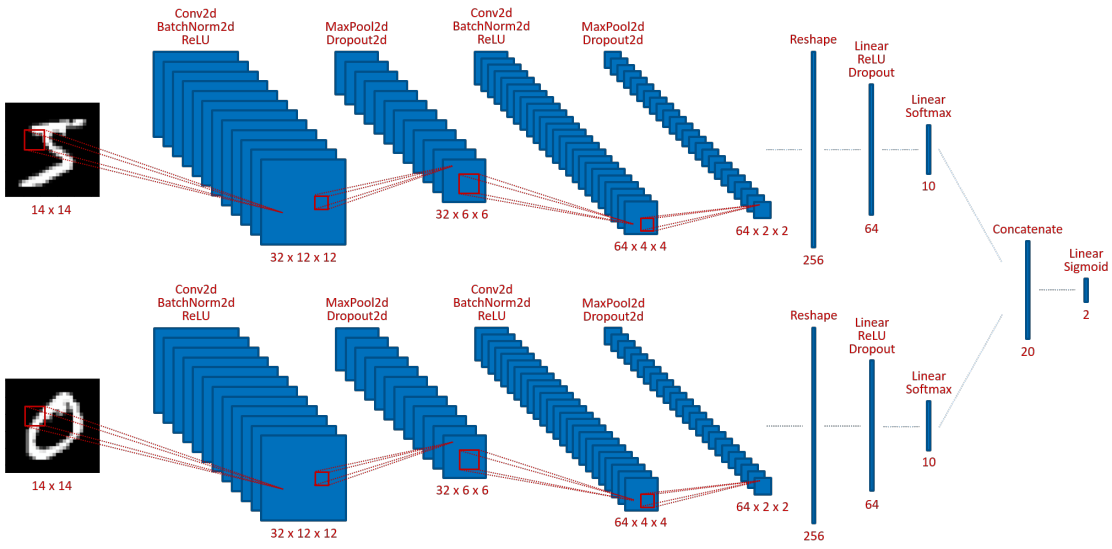

Figure 5: *Auxiliary model architecture*