RDFIA: Pattern recognition for image analysis and interpretation

TP 3/4 & 5/6 Report

Kevin Meetooa

Année 2020-2021

Contents

Chapter 1

TP 3&4

1.1 Introduction

In this report, we present the work done and the results obtained during the TPs 3 and 4.
We first gave a formal definition of a Neural Network, expliciting both its forward and backward passes. We then calculated the gradients with respect to several parameters to get a better understanding of the backpropagation.
After the theorical part, we coded and implemented the neural network.

1.2 Dataset

1.
   - The training set is used to fit the model and its weights.
   - Once the model has been fit on the training set, the validation set is used to tune the model's hyperparameters.
   - Once the model's hyperparameters have been tuned on the validation set, the test set is used to evaluate the performance of the model.

2. A higher value of N means there are more examples in the training set. A higher value of N usually leads to a better chance of getting a model that is able to generalize on larger test sets as long as the training set isn't too imbalanced.

1.3 Neural Network Architecture

3. Activation functions allow us to add some "non-linearity" into the network, this can be better for capturing complex patterns. Without activation functions, our neural network would only be a composition of linear functions, rendering our model unable to treat non-linearly separable problems.

4.
   - $n_x = 2$. This corresponds to the input dimension
   - $n_y = 2$. This corresponds to the number of classes in our problem
   - $n_h = 4$. This is an hyperparameter

5. $\hat{y}$ is a prediction: This is the output of our neural network given an input x. For our classification problem, $\hat{y}$ will be the predicted class for x.

y is the "ground truth": This is the class that we want our neural network to predict.

6. The softmax function allows us to assign a probability to each of the output classes. We can then use this output to choose the class with the highest probability.

7. $\tilde{h} = W_h x + b_h$
   $h = tanh(\tilde{h})$
   $\tilde{y} = W_y h + b_y$
   $y = Softmax(\tilde{y})$

8. In both cases, $\tilde{y}$ must get closer to y to minimize the loss function.

9. MSE is best suited for regression problems because of its sensitiveness to outliers, giving a high penalty to those extreme values.

   CrossEntropy is best suited for classification because it allows us to compute a "distance" between two probability distributions.

1.4 Learning Method

10. - Classical Gradient Descent: This version converges faster (in terms of number of iterations) than the other versions but it's time of execution may be too long for larger datasets.
    - Stochastic Gradient Descent: This version has a much faster execution time but usually converges slower than classical gradient descent.
    - Mini-batch Gradient Descent: This is a good compromise between classical and stochastic gradient descent

    Mini-batch gradient descent seems to be the best choice in the general case. The convergence is usually faster than stochastic gradient descent and the execution time is faster than classical gradient descent.

11. A too small learning rate may lead to a slow convergence whereas a too high learning rate can either lead to a non-optimal solution or even a divergent behaviour.

12. The naive approach would be applying the chain rule on all of the variables used for computing the loss function's gradient. However, using this solution, the same derivatives would be computed several times.

    The backpropagation approach would be computing all the variables' derivatives first and then storing them. We could then apply the chain rule by looking up for the derivatives in the stored table.

    The latter solution is much more computationally efficient.

13. For the backpropagation to work, the loss function must be differentiable.

14. $l(y_i, \hat{y_i}) = -\sum_i y_i log(\hat{y_i})$
    $= -\sum_i y_i log(\frac{e^{\tilde{y_i}}}{\sum_j e^{\tilde{y_j}}})$ (application of the softmax formula)
    $= -(\sum_i y_i \tilde{y_i} - \sum_i y_i log(\sum_j e^{\tilde{y_j}}))$
    $= -\sum_i y_i \tilde{y_i} + log(\sum_i e^{\tilde{y_i}})$ since $\sum_i y_i = 1$ (y is obtained from a softmax)

15. $\frac{\partial l}{\partial \tilde{y_i}} = \frac{\partial}{\partial \tilde{y_i}}(-\sum_i y_i \tilde{y_i} + log(\sum_i e^{\tilde{y_i}}))$

$= -y_i + \frac{e^{\tilde{y_i}}}{\sum_i e^{\tilde{y_i}}}$ since $log(f)' = \frac{f'}{f}$

$= -y_i + \hat{y_i} = (\hat{y_i} - y_i)$

Therefore, $\begin{bmatrix} \frac{\partial l}{\partial \tilde{y_1}} \\ \frac{\partial l}{\partial \tilde{y_2}} \\ \vdots \\ \frac{\partial l}{\partial \tilde{y_{n_y}}} \end{bmatrix} = \begin{bmatrix} \hat{y_1} - y_1 \\ \hat{y_2} - y_2 \\ \vdots \\ \hat{y_{n_y}} - y_{n_y} \end{bmatrix}$

16. $\frac{\partial l}{\partial W_{y,ij}} = \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial W_{y,ij}}$

With $\tilde{y} = W_y h + b_y$

Therefore, $\frac{\partial \tilde{y_k}}{\partial W_{y,ij}} = \frac{\partial}{\partial W_{y,ij}}(\sum_j W_{y,ij} h_j + b_y) = h_j$

And since, $\frac{\partial l}{\partial \tilde{y_k}} = \hat{y_k} - y_k$

Then $\frac{\partial l}{\partial W_{y,ij}} = \sum_k (\hat{y_k} - y_k) h_j = h_j \sum_k (\hat{y_k} - y_k)$

17.

- $(\nabla_{\tilde{h}} l)_i = \frac{\partial l}{\partial \tilde{h_i}} = \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial h_i} \frac{\partial h_i}{\partial \tilde{h_i}}$

  $= \sum_k (\hat{y_k} - y_k) W_{y,ki} (1 - h_i^2)$ since $\frac{\partial \tilde{y_k}}{\partial h_i} = W_{y,ki}$ and $h_i = tanh(\tilde{h_i})$

  $= (1 - h_i^2) \sum_k (\hat{y_k} - y_k) W_{y,ki}$

- $(\nabla_{W_h} l)_{i,j} = \frac{\partial l}{\partial W_{h,ij}} = \frac{\partial l}{\partial \tilde{h_i}} \frac{\partial \tilde{h_i}}{\partial W_{h,ij}}$

  With $\tilde{h} = W_h x + b_h \Rightarrow \frac{\partial \tilde{h_i}}{\partial W_{h,ij}} = x_j$

  Therefore, $\frac{\partial l}{\partial W_{h,ij}} = ((1 - h_i^2) \sum_k (\hat{y_k} - y_k) W_{y,ki}) x_j$

- $(\nabla_{b_h} l)_i = \frac{\partial l}{\partial b_{h_i}} = \frac{\partial l}{\partial \tilde{h_i}} \frac{\partial \tilde{h_i}}{\partial b_{h_i}}$

  With $\tilde{h} = W_h x + b_h \Rightarrow \frac{\partial \tilde{h_i}}{\partial b_{h_i}} = 1$

  Therefore, $\frac{\partial l}{\partial b_{h_i}} = \frac{\partial l}{\partial \tilde{h_i}} = (1 - h_i^2) \sum_k (\hat{y_k} - y_k) W_{y,ki}$

Chapter 2

TP 5&6

2.1 Introduction to convolutional neural networks

1. For an input of shape (x, y, z) and given a convolution filter with padding p, stride s and kernel size k, then the output shape will be:
   $(\frac{x-k+2p}{s} + 1)(\frac{y-k+2p}{s} + 1)$
   For a single filter, there are $k * k * z$ weights to learn since the convolution filter has z inputs.
   A fully-connected layer with the same number of outputs would have $x * y * z * (\frac{x-k+2p}{s} + 1) * (\frac{y-k+2p}{s} + 1)$ parameters.

2. The convolution filter has a much smaller number of parameters compared to the fully-connected filter. The convolution filter is also independent of the shape of the input.
   However, its limit is that it only detects local patterns and not global patterns. This can be solved by taking bigger kernels.

3. A spatial pooling step is usually performed after a convolution. The pooling is used to gradually reduce the shape of the inputs while still keeping the important information. This is used to reduce the number of parameters in the network and also to prevent overfitting.

4. There would be no problem applying the convolution/pooling transformations on an image with a different shape than (224,224) as these transformations are independent of the input shape.
   However, there would be an issue with computing the fully-connected layer since its number of parameters depends on the input shape.

6. If we replace the fully-connected layers by convolutional layers, then there would be no shape issues anymore as the convolutonal layers' number of parameters are independent of the input size.
   However, if the input is bigger than the original input size, then the output will also be bigger than the original output size.

7. Given a convolution filter of kernel size k, then the receptive field shape in the first layer of the network will be $k * k$.
   As we go deeper in the network's layers, the receptive field shape gets higher.

2.2 Model Learning from scratch

8. The output shape n_{out} verifies the following equation:

   $n_{out} = \frac{n_{in}-k+2p}{s} + 1$

   We can solve this equation by substituting n_{out} with n_{in} to obtain the value of s and p.

   $s = \frac{n_{in}-k+2p}{n_{in}-1}$

   $p = \frac{s(n_{in}-1)+k-n_{in}}{2}$

9. We will solve the same equation as above by substituting n_{out} with $\frac{n_{in}}{2}$

   We get $s = \frac{\frac{n_{in}}{2}-k+2p}{\frac{n_{in}}{2}-1}$

   $p = \frac{s(\frac{n_{in}}{2}-1)+k-\frac{n_{in}}{2}}{2}$

Table 2.1: Number of weights for each layer

Layer	Number of weights	Output size
conv1	32 x 5 x 5 x 3 = 2400	32 x 32 x 32 = 32 768
pool1	0	16 x 16 x 32 = 8192
conv2	64 x 5 x 5 x 32 = 51 200	16 x 16 x 64 = 16 384
pool2	0	8 x 8 x 64 = 4096
conv3	64 x 5 x 5 x 64 = 102 400	8 x 8 x 64 = 4096
pool3	0	4 x 4 x 64 = 1024
fc4	4 x 4 x 64 x 1000 = 1 024 000	1000
fc5	1000 x 10 = 10 000	10

10. The table 2.1 shows the number of parameters for each layer. We can see that a very high majority of the weights are located on the first fully-connected layer (fc4).

11. The total number of weights is 2400 + 51 200 + 102 400 + 1 024 000 + 10 000 = 1 190 000

    This number is much higher than the number of training examples (50000).

12. With the BoW/SVM approach, for 1000 words and 10 classes, we would have 1000 * 10 = 10 000 parameters, which is much lower than the number of parameters needed for our current CNN.

14. During the training phase, the weights of the network are updated by performing a back-propagation step on the loss along with an optimization step with the optimizer. During the testing phase, the weights of the network aren't updated.

16. The learning rate and the batch size both have an impact on the speed of convergence of our network.

    A too low learning rate will lead to a slow convergence whereas a too high learning rate may lead to diverging behaviours or non-optimal convergence.

    A small batch size will make the loss function more instable than a high batch size. However, the loss function may converge quicker with a smaller batch size especially when compared to a too big batch size.

17. The error during the first epoch is caused by the random initialization of the weights. It doesn't make sense to interpret it as the model hasn't had the time to correct and optimize its weights yet.

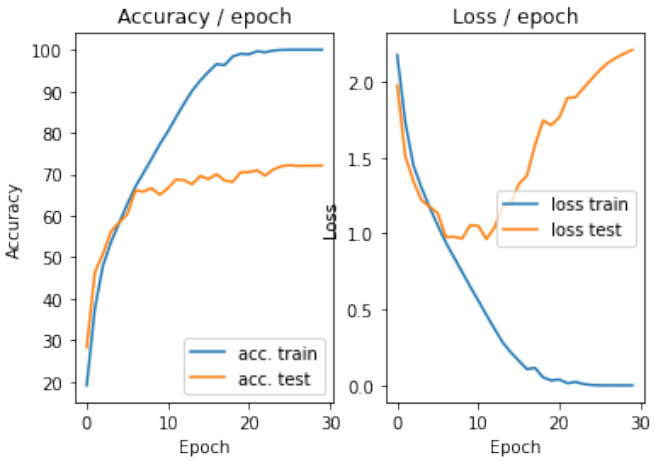

Figure 2.1: First model's performances

18. The test accuracy is much lower than the training accuracy as seen on 2.1. This is an overfitting behaviour: The loss function (on the test dataset) decreases before increasing again.

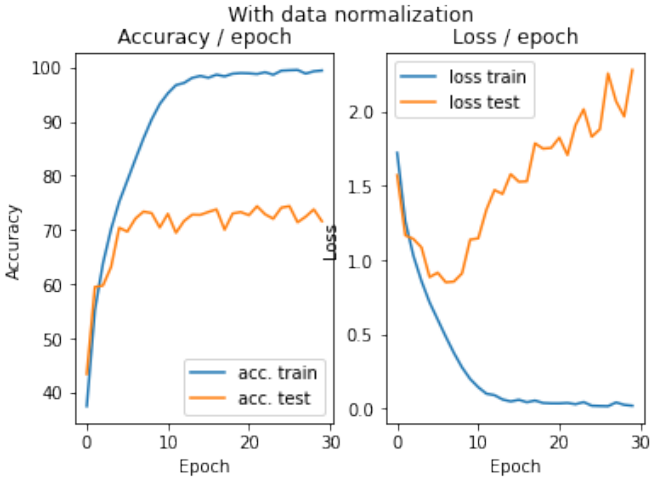

Figure 2.2: Model's performances after data normalization

19. As seen on 2.2, the model's performances have slightly improved (from around 70% in test to around 72%). However, the model is still overfitting.

20. We need to compute the average image by only using the training images. The test and validation sets should only be used for testing our fitted model's performances, not for hyperparameters tuning.

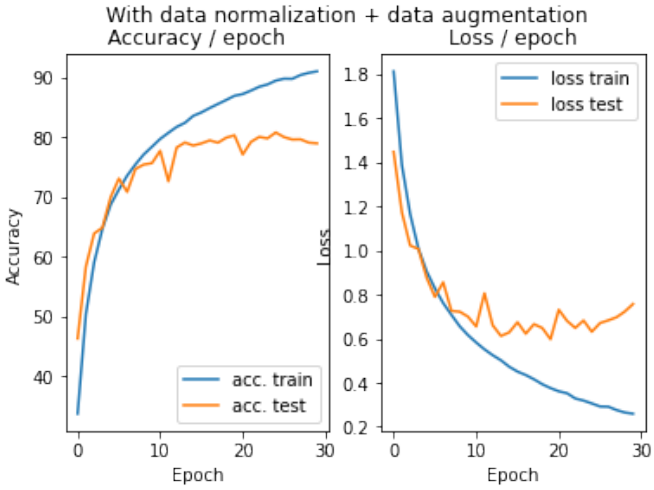

Figure 2.3: Model's performances after data normalization + data augmentation

22. As seen on 2.3, the results are slightly better after performing data augmentation (around 72% without data augmentation and around 78% with data augmentation).

23. The horizontal symmetry approach can't be used on any type of images.

    Some images could lose their meaning after such a transformation, such as images or characters for example. However, it could be used on simple objects.

24. As explained above, there is a risk that some images will lose their meaning.

    These techniques could induce a bias in the dataset and should be used with caution.

25. We could add some noise to the images, perform small rotations/translations or even change light intensity/color saturation.

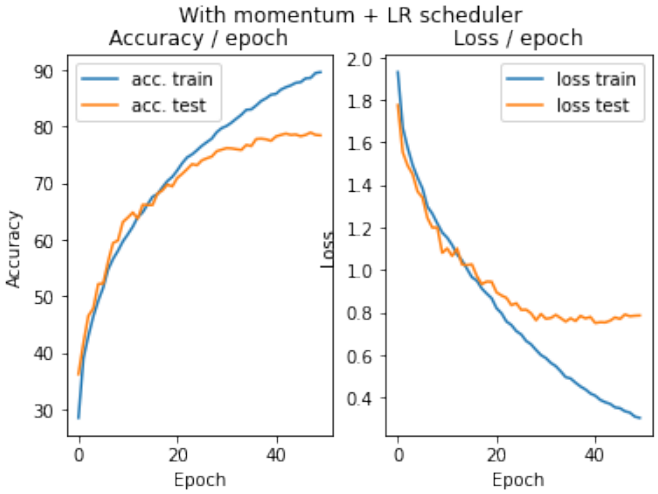

Figure 2.4: Model's performances with data normalization, data augmentation, learning rate scheduler and momentum

26. As seen on 2.4, the results haven't improved when compared to data augmentation only.
    We can see that the loss function is much more stable and less spiky.
    However, the convergence is much slower (around 50 epochs with learning rate scheduler + momentum compared to around 30 epochs without these methods).

27. The momentum sets the weight between the average of previous derivative values and the current value to calculate the new weighted average. Therefore, momentum will help in getting a quicker convergence if we're making a step in the right direction and make the convergence slower if we're making a step in the wrong direction.

    The learning rate scheduler allows us to exponentially decrease the value of the learning rate. This is good for converging towards a good local minimum as we're making big steps at first and smaller steps as we get closer to the minimum.

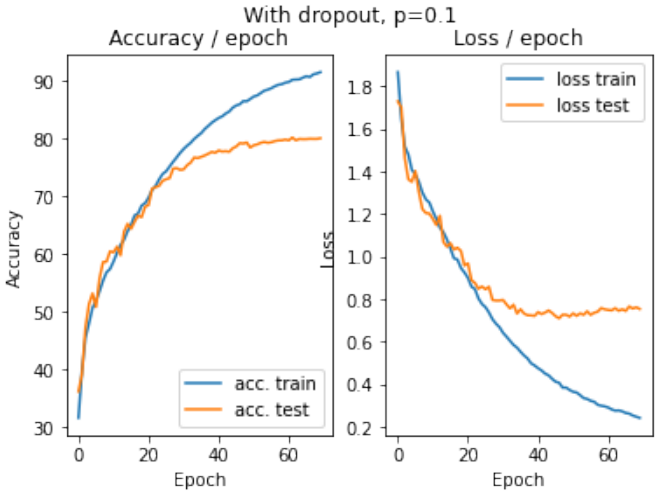

Figure 2.5: Model's performances with data normalization, data augmentation, learning rate scheduler + momentum and dropout

29. The dropout gives much better results as seen on 2.5: Our model isn't overfitting anymore. Therefore, the dropout is a good regularization technique.

30. Regularization is a technique used to reduce overfitting. It usually consists of either lowering some weights (ex: Ridge Regression) or even setting them to 0 (ex: Lasso Regression, Dropout). Therefore, regularization penalizes complex models containing a high number of weights.

31. Dropout "encourages" all the neurons in the network to learn by randomly setting some neurons to 0 in each forward pass.
This will push all the neurons to learn and prevent unwanted behaviour where only a small number of neurons contain most of the information used to perform the classification.

32. The hyperparameter p corresponds to the probability of setting a given neuron to 0. If this hyperparameter is too small, then the dropout's effect won't be noticeable.
However, if it is too high, then the model may be underfitting because of a lack of neurons.

33. The dropout technique is only performed during training for the reasons explained above. During the testing phase, dropout isn't needed as we are testing our network's performances and not tuning its parameters anymore.

34. As seen on 2.6, the results are much better (around 85% accuracy in test) when using batch normalization.
The loss function is much more stable too, especially when compared to previous figures.
We can probably consider the model as satisfying now.

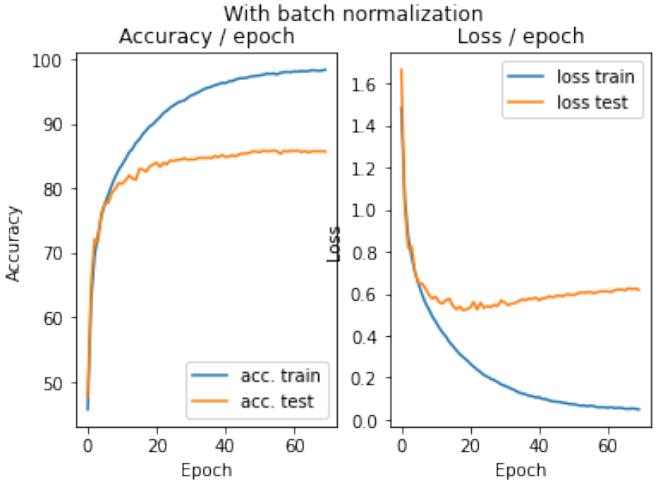

Figure 2.6: Model's performances with data normalization, data augmentation, learning rate scheduler, momentum, dropout and batch normalization