Assignment: Regularization techniques and Autoencoder on Multi-MNIST dataset

David Bertoldi – 735213
email: d.bertoldi@campus.unimib.it

Department of Informatics, Systems and Communication

University of Milano-Bicocca

1 Inspecting the data

The data provided consinst of a set of grey-scale images containing 2-digits handwrittend numbers. The source is a modification of the MNIST dataset: each image is composed by two digits coming from the original MNIST that may be overlapped in different intensity. This let expand the dataset from the set of numbers from 0 to 9 to numbers from 1 to 50.

Figure 1 shows the first 16 samples of the training dataset with unique labels.

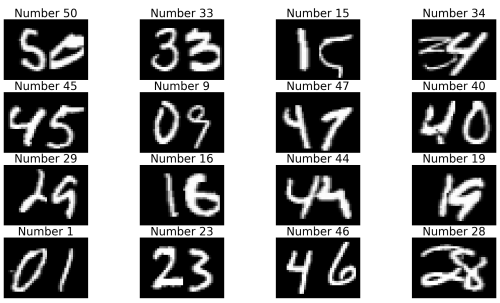

Figure 1: Sample of the first 16 unique samples

The data is already divided in training and test datasets. The first present a non-uniform distribution, as shown in Figure 2. Many numbers have very low occurrencies, like 43 with 117 or 17 with 130, against other like 11 with 3112 or 19 with 2890. As a matter o fact the train dataset has mean $\mu \simeq 1671$ with a standard deviation $\sigma \simeq 1045$, leading to an unbalanced training for some classes.

Along with the bias of the dataset, some number at first sight are difficult to interpret even for the human mind. For example, Figure 3 shows 6 difficult numbers to read.

Before training a FFNN using this images, encoded in 28×39 matrices with values from 0 to 255, we flattened them in arrays 1×1092 and rescaled each value in the continuous interval $[0, 1]$.

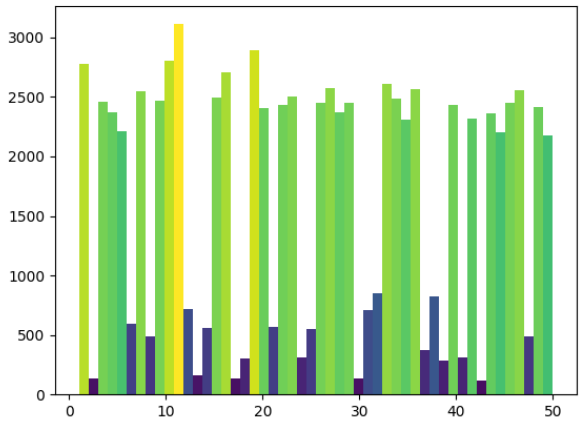

Figure 2: Histogram of the frequency of samples in the dataset

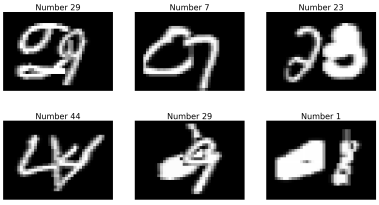

Figure 3: Difficult to read numbers

2 Preparing the data

As said in the section before, the dataset is divided into training and test samples. A validation subset is missing and thus is retrieved from the training set: 20% of the images are randomly used for validation instead of training (along with their labels).

About labels, we encoded them in one-hot vectors so that the 1 are set in the index representing the numerical class.

Another, technical, issue is that `np_utils.to_categorical`, used to create the one-hot

vectors, generates 51 classes instead of 50. That's because the function creates as many classes as the highest value inside the input plus one: it takes for granted that we were using a zero-based index. In order to overcome this, without writing that function from scratch, we subracted 1 to each element of each vector inside the training and test label set. Obvisously that operation is reverted when trying to predict the values.

3 Unregularized FFNN

The aim of this section is to describe a FFNN with less than 100.000 parameters that is able to classify with high level of accuracy the numbers from the dataset without any regularization technique.

3.1 The network

Because the number of parameters are partially determined by the size of the input and output, we tested a FFNN with 2, 3, 4 and 5 hidden layers, but we found that 2-layers model generalized better over a deeper (and less wide) model. Without entering in the details, the deepest model had 5 layers with 70 to 50 neurons each.

We found a good spot with 80 neurons in the first layer and 50 in the seconds, for a total of 96.660 parameters. Figure 4 shows the architecture of the network used in this section.

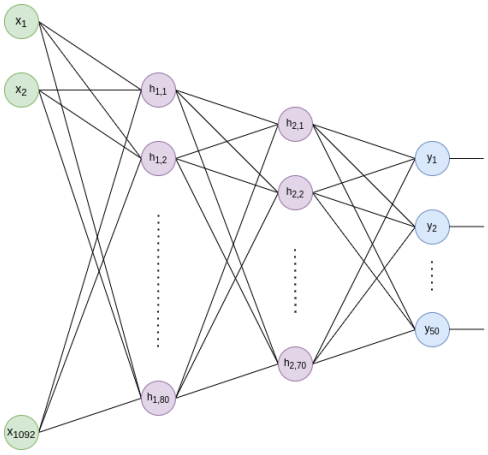

Figure 4: Architecture of the unregularized network

Each hidden neuron uses `Sigmoid` as activation function. We tried `LeakyReLU` with $\alpha = 0.01$ as well but resulting in a divergence in the model (the validation loss kept tp slighty grow after 15 epochs). The source was probably the explosion of the gradient caused by the function.

The output layer computes a `Softmax`, which converts an input vector of real values to an output vector that can be interpreted as categorical probabilities.

3.2 Training

The choice of the optimizer was among *SGD*, *RMSProp* and *Adam*. This selection was influenced also by the initialization of the weights: *SGD* performed well with `GlorotNormal` (Xavier) initializer but not as well as *RMSProp* and *Adam* with `HeNormal`. *SGD* with `HeNormal` resulted in a model that couldn't converge at all.

We tried all of them and chose Adam with learning rate of 0.001 , $\beta_1 = 0.9$ and $\beta_2 = 0.999$ because it seemed to escape better from local minima, converging faster and giving better accuracy.

Because we are trying to find which images best suits in one of the 50 classes, the best loss function is the *Categorical Cross-Entropy*.

The batch size of 256 gave the best results: 512 was another good choice but the convergence was slower and lower values performed worse; that might be due to the fact that the model needed a good amount of variety of information before every update. But even using mini-batches of 8 the validation accuracy reached 89%.

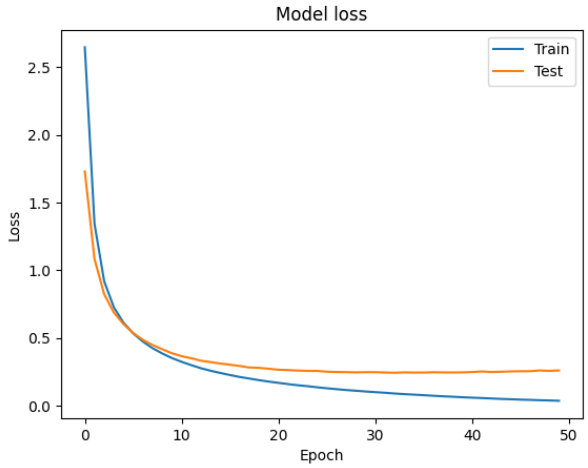

Figure 5: Loss (unregularized)

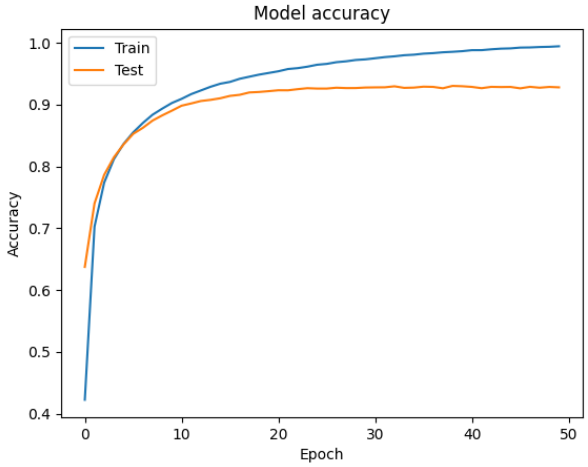

Figure 6: Categorical accuracy (unregularized)

We choose 50 as number of epochs in order to see the effects of missing regularization like *early stopping*, even if the model reached an optimal state after 10 epochs. As shown in Figure 5 and 6 it is possible to see that the training validation reached $\sim$ 100% which means the model overfitted so much that almost memorized the training set. As metric we used the *categorical accuracy* because calculates the percentage of predicted values that match with true values for one-hot labels.

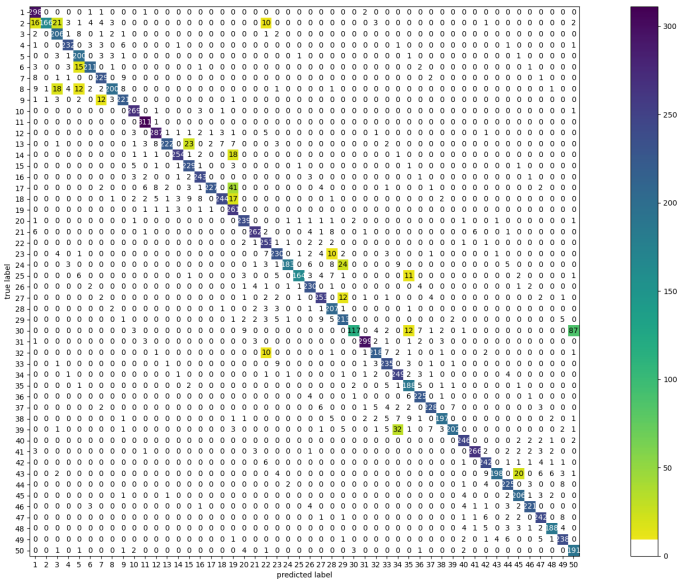

Figure 7: Confusion matrix of the evaluation of the test set (unregularized network)

3.3 Evaluation

The validation accuracy reached 93% after 14 epochs and remained stable till the end. The categorical accuracy over the test set reached 89% and can be analyzed with the help of the confusion matrix (Figure 7). The confusion matrix uses a custom colormap (a *viridis* but with the first 10 values set to 0) so that was easier to hide negligible errors (in this case, less than 10 mismatches). It is noticeable that the model confuses number 50 with 30, 19 with 17, 34 with 39 and others. These errors are not only due to the similarity between their digits, but because these are the classes with less elements in the training set. So with no surprise the unbalances inside the training set played an important role.

4 Regularized FFNN

The aim of this section is to describe a FFNN with less than 100.000 parameters that is able to classify with high level of accuracy the numbers from the dataset with one or more regularization technique.

4.1 The network

In order to better measure the effects of regularization we used the very same architecture as base (2 layers, 80 and 70 neurons).

4.2 Regularization techniques

In this section we describe the techiques tested, used or discarded to achieve a lower level of underfitting and secondary to reach a better level of accuracy.

4.2.1 Data augmentation

The first attempt of undirect regularization involved filling the gap between the classes by generating new samples for the less populated classes. The procedure, described in Algorithm 1, generates for each class as many samples as there are in the percentage p of the difference with the most populated class. By choosing $p = 10\%$ each class diminuishes the gap with the major class by 10%.

Three techniques were taken in account: *adding noise*, *image shifting* and *image rotation*. The first added a gaussian noise $\mathcal{N}(0, \frac{1}{2})$, the second randomly shifted the image along the 2 axis by 2 in both directions and the third rotated the image by a random angle bewteen $-20°$ and $20°$.

None of the above helped the network: the level of accuracy dropped to 86% and the loss was very high when applying noise ($\geqslant 0.6$). Image roation alone is the only one that didn't make the accuracy worse. For these reasons data augmentation was discarded even if theoretically speaking filling the gap between samples makes sense. Probably the model already recognized single features for those samples so that adding similar samples just increased redundancy.

4.2.2 L1 and L2

L1 and *L2*

4.2.3 Dropout

Dropout turned out to be a good choice: it dropped the overfitting without losing too much accuracy. We chose a drop out probability for each layer of 10% because the network wasn't too big and higher probabilities gave worse results. This strategy helped the network to ignore certain features of the images or the $0s$ of the matrices (black spaces). A representation of the network can be found in Figure 8.

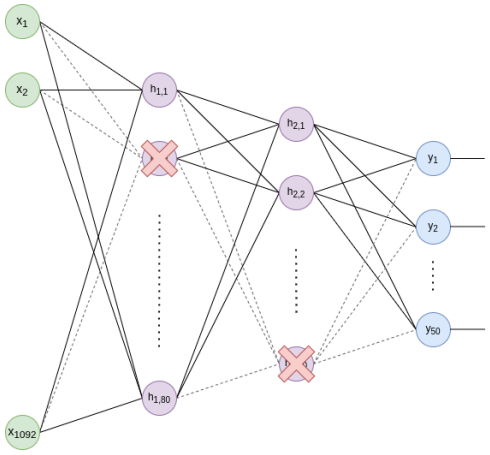

Figure 8: Architecture of the network with dropout

4.2.4 Early stopping

Another game changer was the application of *early stopping* on the validation, with a patience of 10 epochs and $\delta = 0.05$. This greately reduced the overfitting and because the model converged after 15-20 epochs we restored the weights to the last best snapshot (20^{th} epoch).

4.3 Evaluation

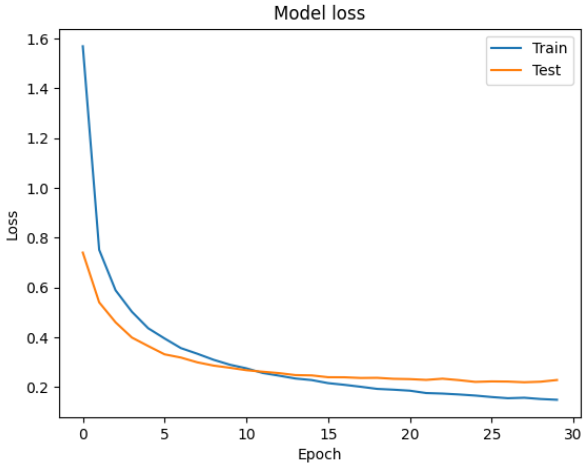

Figure 9: Loss (unregularized)

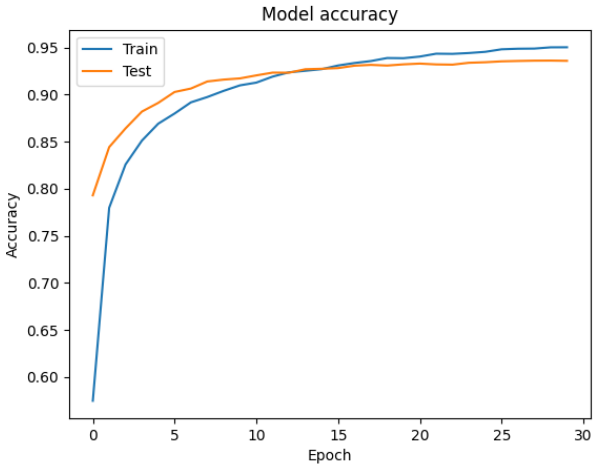

Figure 10: Categorical accuracy (unregularized)

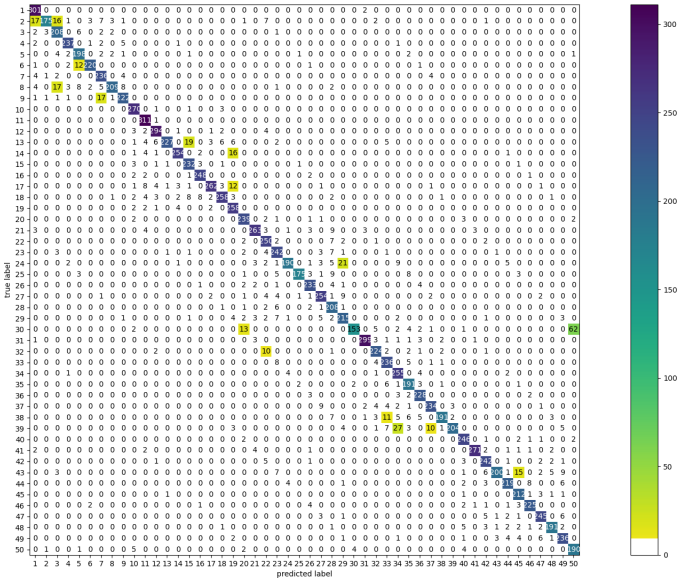

Figure 11: Confusion matrix of the evaluation of the test set (regularized network)

Algorithm 1: Data augmentation algorithm

Data: X *training samples*, Y training labels, $p \in \mathbb{N}$
Result: X, Y
$l = |Y|$;
$C \leftarrow \{\}$;
for $i \leftarrow 0$ ***to*** l **do**
  if $Y_i \in C$ **then**
    $C_i \leftarrow C_i + 1$;
  else
    $C_i \leftarrow 1$;
  end
end
$m \leftarrow \max_k C_k$;
for $(k, v) \in C$ **do**
  $S = \{e \in X : e = k\}$;
  $g \leftarrow \left\lfloor \frac{m-v}{p} \right\rfloor$;
  for $i \leftarrow 0$ ***to*** g **do**
    $x \leftarrow augment(S_{random})$;
    append x to X;
    append k to Y;
  end
end
return $shuffle(X, y)$;