# WASC – Deep Neural Network

Walter Zigliotto, Andrea Nicolai, Sandeep Kumar Shekhar, and Camilla Quaglia

(Dated: April 16, 2020)

DDN is ...

## INTRODUCTION

The main purpose of this project is to setup a Deep Neural Network with only two layers in order to see which set of parameters leads to the best performance. The goal is to predict whether a **SEE AGAIN VIDEOS FOR NUMBERS** sequence of 7 numbers hides a key **OR TWO?** subsequence of **XX** digits inside itself, once we have trained the network on a provided dataset.

WHAT IS A DNN.

just few lines on parameters we changed: WHAT IS AN ACTIVATION FUNCTION (it must be not linear, it rules output of single nodes...)
WHAT IS A REGULARIZATION (regularize weights in order to prevent overfitting)
WHAT IS A DROPOUT (we "burn" some nodes in order to enlarge stochasticity and thus generalization "property" of the network therefore preventing overfitting)
WHAT IS AN OPTIMIZER (to train network we minimize the loss wrt to weights at each epoch, so in order to find the minimum we use different algorithms)

## METHODS

To create and train our NN we used Tensorflow 2.1.0 that already contains Keras API implemented. **REFERENCE TO TENSORFLOW DOCUMENTATION AND BELOW TO KERAS**. Moreover, we did try also a different configuration (Keras API v. 2.2.4 using Tensorflow v. 1.14.0 backend) thus leading to different results, despite the use of the same set of best parameters.
As previously stated our training set constitued by a sequence of 7 digits that might contain a couple of subsequences of two digits each. If so, then the correct label would be 1, thus becoming a problem of *binary classification*. Each possible digit was indeed in the set $D \in 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$: therefore it fixed some constraints of the shape of the first layer. Since each number in $D$ could be present in any of the 7 position of our sequence, the *input* layer needed to be composed by exactly $D * L = 63$ nodes. Moreover, following the instructions that were provided, we had to obey to the constraints that the *first* and *second* layer must be formed, at most, respectively by $(LD/2, LD/4)$ vertices.
In order to find the best set of parameters, we implemented a CV grid search, using 80% of the whole dataset as a training set thus splitting it into 10 folds, and using a batch size of 50. Training last 40 epochs, reshuffling the set after each single iteration. Moreover for the whole research, we set the seed 1234. Finally, after having obtained the best set of parameters, we try to predict the labels of the remaining 20% dataset, e.g. the so-called *test set*.
Since parameters to be tuned were only four, we thought that a good compromise would be using just only a couple of CV Grids Search: the first one would allow us to find the best *Optimizer*, *Regularizer (Ridge)* parameters, while the last one *activation function* and *Drop out* rate. All paramaters were moreover to be set equal for all the layers, except for the last before the *output* one, whose nodes were burnt out according to the Drop out rate.
All our work consisted in applying different transformations to our dataset, and see whether it may lead to some improvements in the performance of our trained NN, always by the mean of Grid Search with Cross Validation. We expected to see that for different transformations, would correspond different set of parameters.
First we used the "smooth" dataset as it had been delivered to us. It was made up by 3000 entries. We expected to fall into some overfitting problems, and actually it happened as it will be shown in the results section.
Secondly we tried, once we had reshaped our data in a way compatible to feed the Network, to shift and rescale them. This meant that, for each of the $L * D = 63$ mentioned before, we subtracted its mean thus normalizing in order to set its standard deviation to 1. After having applied this transformation, we used Grid Search CV.
Lastly, we noticed that in our dataset there was an invariance of the results with respect to shift of the digits. This could be translated in the fact that if a certain sequence (as an example 345**67**89) contained the key, then also the $L-1$ equivalent sequences would hold it (45**67**893, 9345**67**8 and so on...). In order to augment our dataset, we defined the following function that accepts in input the data and its labels:

```
def expand_augment(data, label):
  S = data

  #small debug
  if(len(str(S))!=L):
    print('mismatch!')
    return[]
```

```
#initialize empty vectors
x_temp = [0] * LD
y_temp = label

p = 10**(L-1)
j = 0

#in this way we obtain the first digit (MSD)
while (j < L):
  q = int(S/p)

  # 1...9 ---> 0...8
  x_temp[j*D + (q-1)] = 1
  j += 1
  S = S - q*p
  p = int(p/10)

x_aug  = [0] * LD * L
x_aug  = np.reshape(x_aug, (L, LD))
y_aug  = np.array([0]*L)

#for each combination possible by shifting
for combination in range(L):
  #for each possible number
  for index in range(LD):
    x_aug[combination][(index +
    ↪  combination*D)%LD] = x_temp[index]
    y_aug[combination] = label

return x_aug, y_aug
```

In this way we were able to obtain a larger dataset by a factor $L-1=6$. As before, we trained our NN using a Cross Validation Grid Search in order to find the set of parameters that best predict the labels of the test set.

## RESULTS

HOW DIFFERENT METHODS WORKED: -FOR NORMAL DATA WE HAVE SMALL ACCURACY AND OVERFITTING (PLOT)
-FOR AUGMENTED DATA WE HAVE PERFECT PERFORMANCE
-RESHIFT IS JUST A WAISTE OF TIME

WHICH SET OF PARAMETERS HAS LED US TO WIN THE COMPETITION

A FEW LINES ABOUT TENSORFLOW BACKEND THAT IF 2.0 WE OBTAIN 100% PERFORMANCE, WITH SAME SAME SET OF PARAMETERS (even seed) IT IS DIFFERENT.

## CONCLUSIONS

PRAISE THE SUN