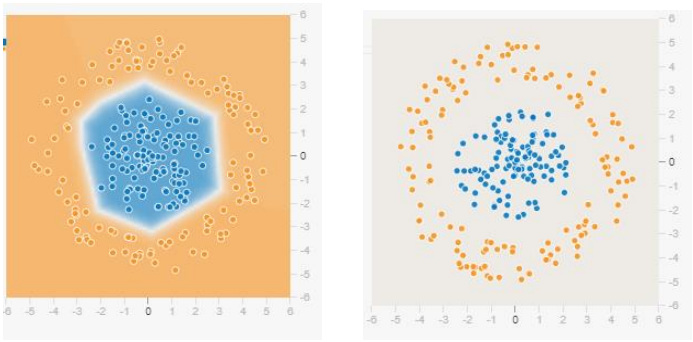
Machine Learning & Data Mining Diego Causarano

Set 4 February 7th 2023

CS 155 PS4

1 Deep Learning Principles [35 Points]

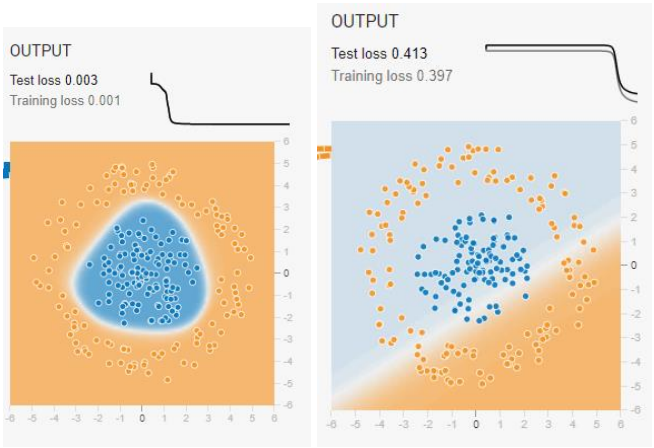
Problem A [5 points]:



First demo Second demo

The difference between the first and the second neural networks is that the first one is initialized with random weights, meanwhile the second neural network is initialized with weights equal to 0. Considering the backpropagation formula, this means that for the second neural network, it will cause the neurons to perform the same calculation in each iteration and produce the same output, because the derivatives ds^L/dx^{L-1} will remain the same for every w^L, thus there will be no learning in each iteration. Also, we know that te ReLU – max(0,S) will saturate, and gradients will vanish in layers after the first one.

Problem B [5 points]:



First demo Second demo

As we can see, the decision boundary for the sigmoid looks smoother than the decision boundary given in the previous item using ReLU. We know that the sigmoid is always differentiable, unlike ReLU.

For the first model, when using sigmoid function, the model learns much slower than the ReLU (Loss function converges in around 200 epochs for ReLU meanwhile it takes around 2000 for sigmoid to stabilize in its final value). This is because the sigmoid is a saturating non-linearity function, which means the derivatives are small, and the product of many small derivatives in the backprop formula goes to zero (as we have seen on the lecture). For the sigmoid, the derivatives goes to zero at -inf +inf. This finally causes that the weights are updated very slowly.

in backprop, the product of many small terms (i.e. $\frac{\partial \mathbf{x}^{(\ell)}}{\partial \mathbf{s}^{(\ell)}}$) goes to zero

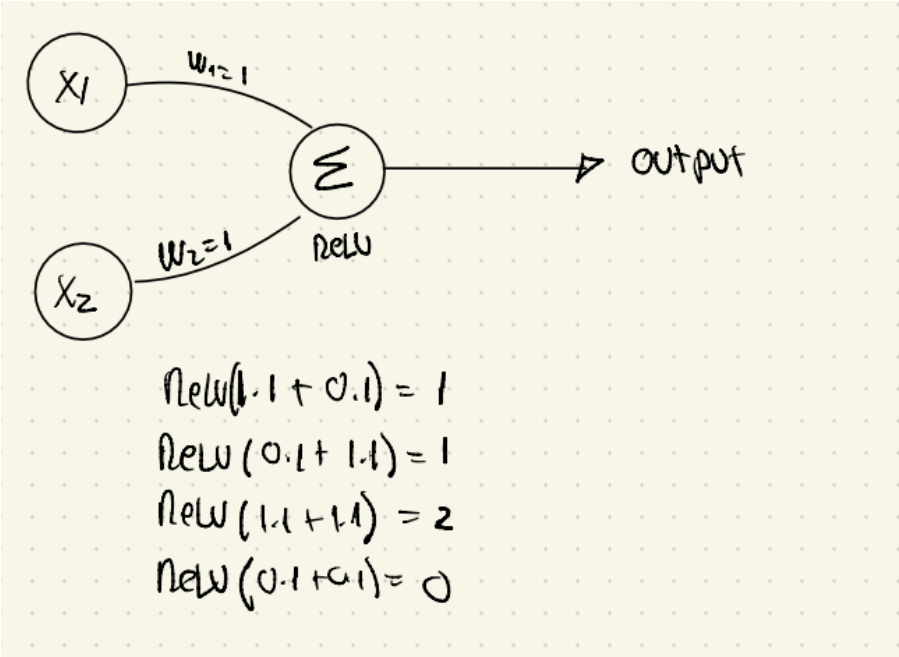
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(\ell)}} = \cdots \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \cdots \frac{\partial \mathbf{x}^{(L-1)}}{\partial \mathbf{s}^{(L-1)}} \cdots \frac{\partial \mathbf{x}^{(\ell+1)}}{\partial \mathbf{s}^{(\ell+1)}} \cdots \frac{\partial \mathbf{x}^{(\ell)}}{\partial \mathbf{s}^{(\ell)}} \frac{\partial \mathbf{s}^{(\ell)}}{\partial \mathbf{W}^{(\ell)}}$$

For the second model, around epoch 3300 for sigmoid starts to learn something, ending with a loss of around 0.4 and with a linear boundary that classifies a little bit better than flipping a coin (for the ReLU case the model didn't learn anything). Again, this is because for the sigmoid function, the gradients are small but non zero, and even when the weighs are initialized as zero, after many epochs the model starts to learn something by using the backpropagation algorithm.

Problem C: [10 Points]

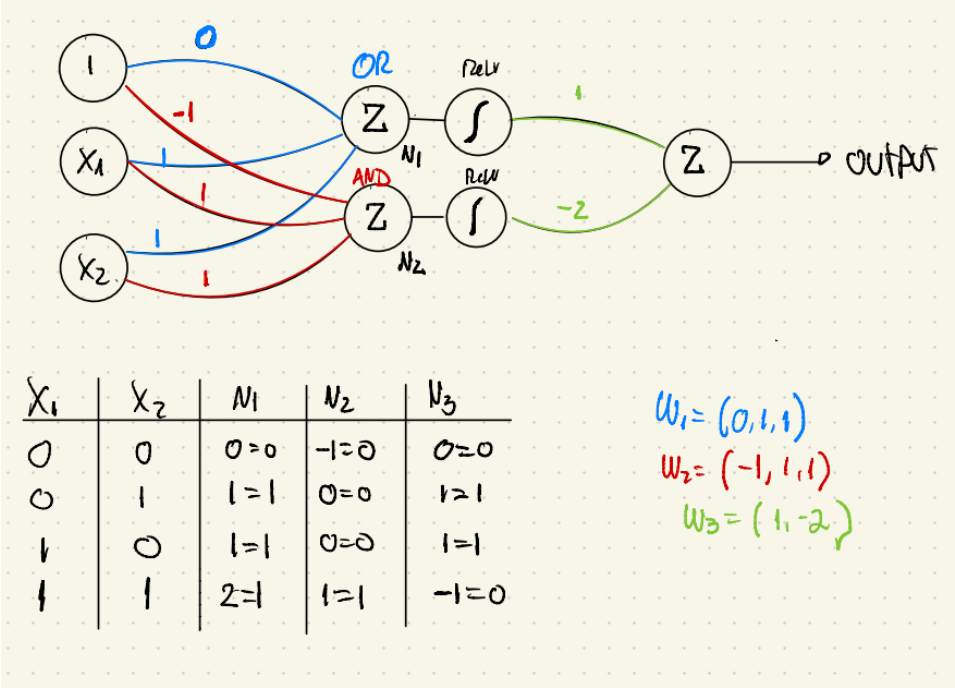
The dying ReLU problem occurs when neurons output a value of zero, and this happens when the input is negative (as mentioned in the exercise). This occurs because when most of the neurons output are zero (for example in this particular case we first feed the model with all the 500 negative points, producing outputs of zero) the model get adjusted for these negative points to have a correct classification, and then, when we start to feeding the model with the positive points as inputs, they will be misclassified as negative. When most of the outputs are zero (the first 500 points), the gradient can not flow and the weights are not being updated, which means that the model stops learning, and as the slope of the ReLU functions goes to zero, once it dead it is impossible to recover the network to learn.

Problem D: Approximating Functions Part 1 **[7 Points]**



Problem E: Approximating Functions Part 2 **[8 Points]**

The minimum number of fully connected layers is two; one hidden layer and one output layer. The explanation is that each layer can only compute one linear function, when the XOR is a non linear function. For this reason, we need to make a linear combination of the OR and AND. In the first layer we will have the OR and AND, in order to substract the case of (x1,x2)=(1,1) that gives and output 1 in the OR and in the AND function, and with this have the desired output.



X_1	X_2	N_1	N_2	N_3
0	0	0≥0	-1≥0	0≥0
0	1	1≥1	0≥0	1≥1
1	0	1≥1	0≥0	1≥1
1	1	2≥1	1≥1	-1≥0

$W_1 = (0, 1, 1)$

$W_2 = (-1, 1, 1)$

$W_3 = (1, -2)$

2 Inside a Neural Network [17 Points]

https://colab.research.google.com/drive/1hb7S8JELgG7teGZNOVatkFhzJa9Ts1fM?usp=sharing

Problem A: Installation [2 Points]

Pytorch versión: 1.13.1

Torchvision: 0.14.1

Problem B: The Data [5 Points]

First, I dropped all the "missing" and "unknown values in the death_yn column because this option was no optional. Then I decided to change the format of the columns with dates to a numeric format in order to have these values in number format, but finally I decided to drop these four entire columns (related to dates) because I think they are not very important for the outcome we are trying to predict (death_yn). Then, for columns with qualitative data such as "current status" , "sex", "Age group", "race ethnicity combined" and columns with binary information such as "hosp_yn", "icu_yn", "medcond_yn and "death_yn", I factorized them in order to have this values in numeric for and being able to train them. After dropped the death_yn with "missing" and unknown" values, I decided to not drop more rows with "missing" or unknown" values in other columns in order to not loose more information, and just treat them as another "factor".

Problem C: Linear Neural Network [5 Points]

Code in google colab link

Problem D: 2-Layer Linear Neural Network [5 Points]

Code in google colab link.

It is expected that 1-;ayer and 2-layer neural networks have similar losses because when there is no activaction function (such as ReLU), the model is just a linear model, regardless of its number of layers.

3 Depth vs Width on the MNIST Dataset [23 Points]

Problem A: The Data [3 Points]

The height and width are 28. The values on each array index represent the intensity of each of the pixels in the image. There are 60,000 images in the training set and 10,000 images in the testing set.

Problem B: Modeling Part 1 [8 Points]

https://colab.research.google.com/drive/1Xad0qfxP2VLVKA8GQkswYkuOisuR9qo6?usp=sharing

Problem C: Modeling Part 2 [6 Points]

https://colab.research.google.com/drive/1_-dRMhPdyWugaRUlQTLobLSDe-VBOLJ7?usp=sharing

Problem D: Modeling Part 3 [6 Points]

https://colab.research.google.com/drive/1Q9WPNivE9MkEFxNn6wBYvT_r3bVYs1pw?usp=sharing

4 Convolutional Neural Networks [40 Points]

Problem A: Zero Padding **[5 Points]**

A pro of zero padding is that it allows to preserve the original size of the input in the output and with this, preserve the highest amount of information as possible. This allows more accurate analysis of images. If we have many deeper networks, padding would cause the reduction of the size in each convolution and the information of the edges will lost.

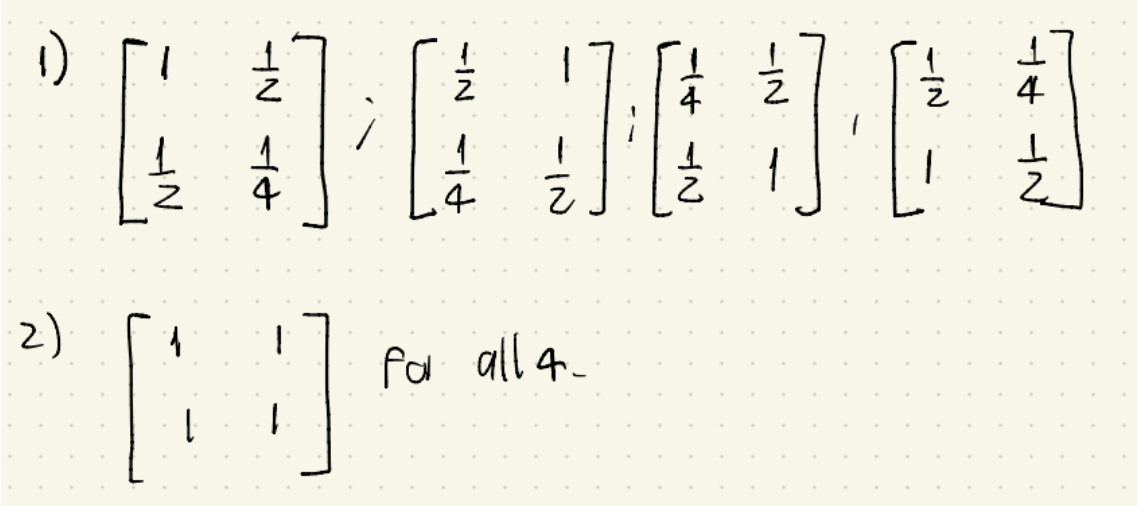
A drawback of zero padding is that the model need to scan a higher number of points, which is translated in a higher amount of computational resources needed. Another disadvantage is that artificial discontinuities are introduced at the borders

Problem B [2 points]:

The number of parameters (weights) in this layer is (5*5*3+1)*8 = 608

The shape of the output tensor is 28x28x8

Problem C [3 points]:



Problem D [4 points]:

Using pooling will avoid the model to "learn" that this small distortions are patterns from the training set. Considering that pooling will take the average of a grid, noises on this wont affect very much to the model, so if this noise is something rare (as mentioned in the exercise) in each grid, the model wont be very sensitive to them, and thus, the final outcome wont be affected so much by them.

Problem E [5 points]:

Grid Search

Pros: Is easy to implement and with this method we will be able to find the best hyper-parameters for the model in the grid .

Cons: It uses a lot of computational resources. Considering that it iterates through all the possible combinations. This means that is computational expensive and takes more time. Also, it iterates without considering past experiences, something that can lead to save resources.

Problem F [20 points]:

The tips in the instructions were very useful. I started with the code sample and then I was adding combinations of changes in the hyperparameters to see the differences. At the beginning I added many layers because my intuition was that more hyperparameters and layers would directly result in better accuracy, but it was not the case. Then I tried to put many convolutional layers with 8 inputs and out puts (all of them the same) and it barely increase my final accurate. Then I tried using many linear layers after the convolutional, changing the numbers of parameters in each layer but it also didn't give me the expected accuracy. Then, reading about some tips in how to improve the final accuracy, I designed a cnn with just two convolutional layers, where the outputs where increasing in each layer (factor of two) to then pass to a layer of just one linear (the output layer) which increased the accuracy but still it wasn't the expected 0.985. Finally, was adding the dropout and the batch normalization in the transition between the convolutional layer and the linear layer the change that made my model to reach the expected accuracy.

https://colab.research.google.com/drive/1j0k_VepAdYqurSSyvTnF6HCnacpCP5c6?usp=sharing

test accuracy for the 10 dropout probabilities *p* [0, 1]

```
Epoch 1/1:, Dropout Prob: 0.0
...........
       loss: 2.4344, acc: 0.1308, val loss: 2.4032, val acc: 0.1371
Epoch 1/1:, Dropout Prob: 0.1
...........
       loss: 2.4575, acc: 0.0812, val loss: 2.4381, val acc: 0.0837
Epoch 1/1:, Dropout Prob: 0.2
...........
       loss: 2.3386, acc: 0.1914, val loss: 2.3213, val acc: 0.2004
Epoch 1/1:, Dropout Prob: 0.30000000000000004
...........
       loss: 2.4409, acc: 0.1609, val loss: 2.4249, val acc: 0.1657
Epoch 1/1:, Dropout Prob: 0.4
...........
       loss: 2.3273, acc: 0.1701, val loss: 2.3158, val acc: 0.1717
Epoch 1/1:, Dropout Prob: 0.5
...........
       loss: 2.3387, acc: 0.1061, val loss: 2.3159, val acc: 0.1089
Epoch 1/1:, Dropout Prob: 0.6000000000000001
...........
       loss: 2.4649, acc: 0.0878, val loss: 2.4365, val acc: 0.0898
Epoch 1/1:, Dropout Prob: 0.7000000000000001
...........
       loss: 2.4233, acc: 0.0991, val loss: 2.4133, val acc: 0.1038
Epoch 1/1:, Dropout Prob: 0.8
...........
       loss: 2.5383, acc: 0.0848, val loss: 2.5365, val acc: 0.0786
Epoch 1/1:, Dropout Prob: 0.9
...........
       loss: 2.4847, acc: 0.0653, val loss: 2.4789, val acc: 0.0569
```

final test accuracy when your model is trained for 10 epochs

```
Epoch 1/10:...........
        loss: 0.1689, acc: 0.9575, val loss: 0.0615, val acc: 0.9808
Epoch 2/10:...........
        loss: 0.0660, acc: 0.9800, val loss: 0.0427, val acc: 0.9872
Epoch 3/10:...........
        loss: 0.0527, acc: 0.9847, val loss: 0.0734, val acc: 0.9863
Epoch 4/10:...........
        loss: 0.0456, acc: 0.9871, val loss: 0.0543, val acc: 0.9832
Epoch 5/10:...........
        loss: 0.0422, acc: 0.9885, val loss: 0.0329, val acc: 0.9904
Epoch 6/10:...........
        loss: 0.0391, acc: 0.9896, val loss: 0.0373, val acc: 0.9903
Epoch 7/10:...........
        loss: 0.0381, acc: 0.9899, val loss: 0.0482, val acc: 0.9896
Epoch 8/10:...........
        loss: 0.0355, acc: 0.9912, val loss: 0.0479, val acc: 0.9901
Epoch 9/10:...........
        loss: 0.0344, acc: 0.9910, val loss: 0.0579, val acc: 0.9880
Epoch 10/10:...........
        loss: 0.0334, acc: 0.9918, val loss: 0.0457, val acc: 0.9921
```