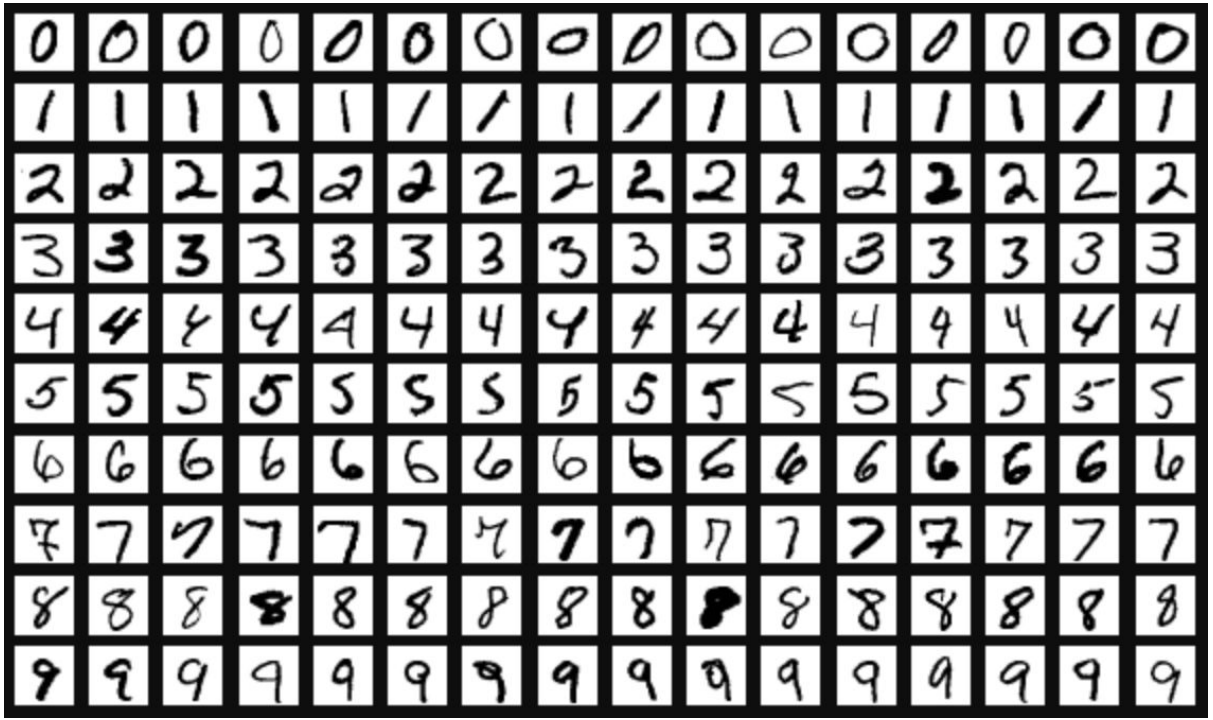
Redes neuronales

Alejandro Garcia
Juan Plúa

Introducción

1. Analizar los datos que estamos utilizando.
2. Explicación medidas de rendimiento.
3. Resultados con distintos valores en los hiperparámetros.
4. Resultados distintas arquitecturas.
5. Normalización.
6. Inicialización.

MNIST



Accuracy

Valores reales: 0 0 0 1 0 0 0 0 1 0 0

Valores predecidos: 0 0 0 0 0 0 0 0 0 0 0

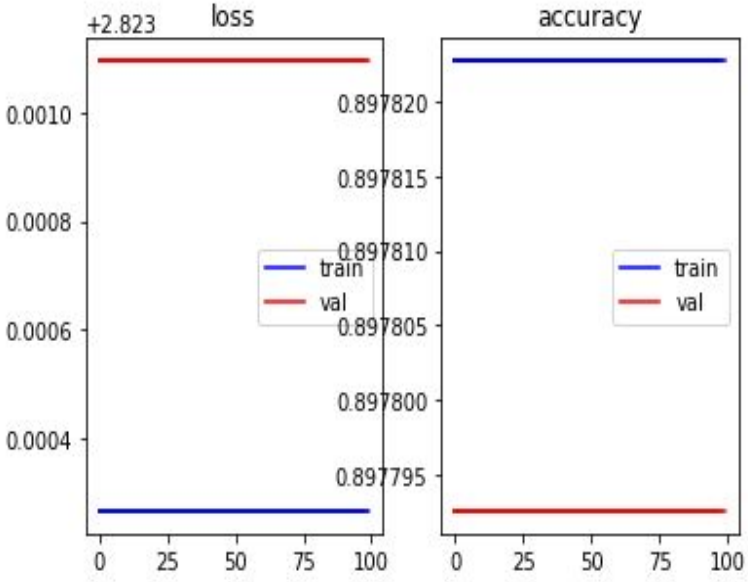
True Negative = 9

False Negative = 2

Accuracy = 9/11 = 0.82

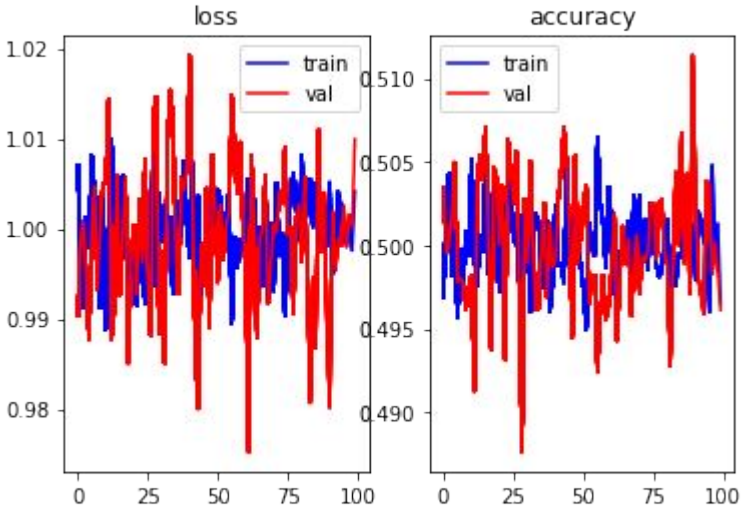
Nos interesa darle importancia al los True Positive.

Accuracy



Accuracy con un modelo que devuelve siempre 0

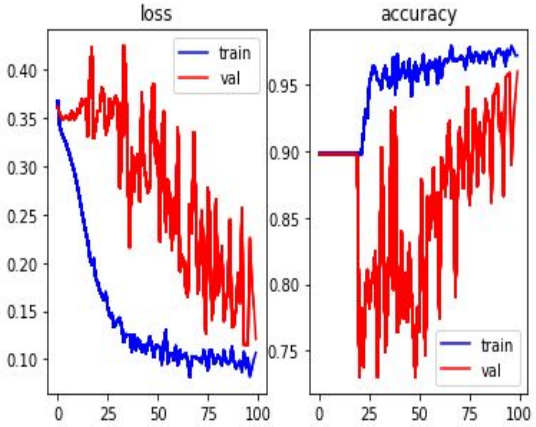
Accuracy



Accuracy con un modelo que devuelve valores random entre 0 y 1

Resultados diferents hiperparámetros

Hyperparameter					Loss	
N. Prueba	Momentum	Learning rate	batch size	epochs	training	validation
1	0.5	0.07	100	100	0.1	0.65
2	0.1	0.07	100	100	0.12	0.21
3	0.1	0.01	100	100	0.09	0.42
4	0.1	0.05	100	100	0.1	0.45
5	0.1	0.1	1000	100	0.19	0.16
6	0.1	0.1	1000	10	0.30	0.34
7	0.1	0.000001	1000	10	0.84	0.84
8	0.1	0.07	1000	100	0.11	0.13
9	0	0.1	1000	100	0.1	0.12



Resultados distintas arquitecturas

Architectures						
# layers	Layer	# neurons	Accuracy		Loss	
			Training	Validation	Training	Validation
3	Layer 1	28^2	0.9600375	0.7934194	0.1126689	0.3824305
	Layer 2	4				
	Layer 3	4				
4	Layer 1	28^2	0.9734764	0.7059558	0.0862636	0.5263381
	Layer 2	4				
	Layer 3	4				
	Layer 4	4				
5	Layer 1	28^2	0.9732055	0.4119950	0.0846971	1.0026668
	Layer 2	4				
	Layer 3	4				
	Layer 4	4				
	Layer 5	4				
2	Layer 1	28^2	0.9289717	0.1022074	0.1979476	1.7746183
	Layer 2	4				
3	Layer 1	28^2	0.9809355	0.7556851	0.0762669	0.7112880
	Layer 2	10				
	Layer 3	10				
3	Layer 1	28^2	0.9885821	0.5063915	0.0377745	1.8137933
	Layer 2	32				
	Layer 3	64				
4	Layer 1	28^2	0.9928325	0.4370678	0.0244635	1.8945795
	Layer 2	50				
	Layer 3	50				
	Layer 4	50				

Fijar weights y learning rate

```python
#init weights
def init_weights(m):
    if type(m) == torch.nn.Linear:
        m.weight.data.fill_(0.01)
```

```python
#Binary output
#Instantiate network
model = NeuralNet()
model.apply(init_weights)
model.state_dict()
```

```
OrderedDict([('layer1.weight',
              tensor([[0.0100, 0.0100, 0.0100,  ..., 0.0100, 0.0100, 0.0100],
                      [0.0100, 0.0100, 0.0100,  ..., 0.0100, 0.0100, 0.0100],
                      [0.0100, 0.0100, 0.0100,  ..., 0.0100, 0.0100, 0.0100],
                      [0.0100, 0.0100, 0.0100,  ..., 0.0100, 0.0100, 0.0100]])),
             ('layer1.bias', tensor([ 0.0339, -0.0337,  0.0317, -0.0268])),
             ('layer2.weight', tensor([[0.0100, 0.0100, 0.0100, 0.0100],
                      [0.0100, 0.0100, 0.0100, 0.0100],
                      [0.0100, 0.0100, 0.0100, 0.0100],
                      [0.0100, 0.0100, 0.0100, 0.0100]])),
             ('layer2.bias', tensor([-0.2732,  0.3924,  0.2143,  0.4799])),
             ('output.weight', tensor([[0.0100, 0.0100, 0.0100, 0.0100]])),
             ('output.bias', tensor([-0.4550]))])
```

Resultados distintas arquitecturas

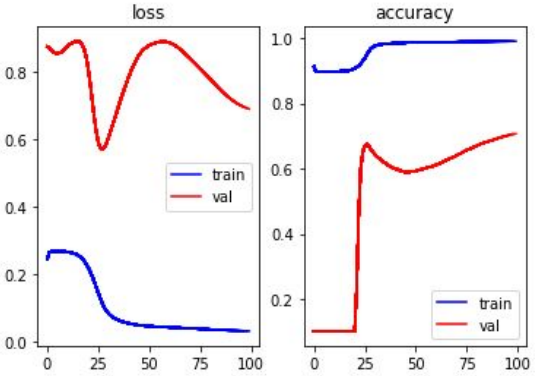
Architectures							
# layers	Layer	# neurons	Activation function	Accuracy		Loss	
				Training	Validation	Training	Validation
3	Layer 1	28^2	Sigmoid	0.897822	0.8977925	0.335947	0.3401864
	Layer 2	4					
	Layer 3	4					
4	Layer 1	28^2	Sigmoid	0.897822	0.8977925	0.335953	0.3406793
	Layer 2	4					
	Layer 3	4					
	Layer 4	4					
4	Layer 1	28^2	Sigmoid	0.897822	0.8977925	0.336695	0.3491937
	Layer 2	10					
	Layer 3	10					
	Layer 4	10					
3	Layer 1	28^2	Tanh	0.98291	0.871303	0.05447	0.276374
	Layer 2	4	Tanh				
	Layer 3	4	Sigmoid				
4	Layer 1	28^2	Tanh	0.972726	0.891461	0.085433	0.246182
	Layer 2	50	Tanh				
	Layer 3	50	Tanh				
	Layer 4	50	Sigmoid				
5	Layer 1	28^2	relu	0.98181	0.864639	0.06610	0.28013
	Layer 2	50	relu				
	Layer 3	50	relu				
	Layer 4	50	Sigmoid				
3	Layer 1	28^2	relu	0.97837	0.87788	0.07526	0.25703
	Layer 2	4	relu				
	Layer 3	4	Sigmoid				

Normalización

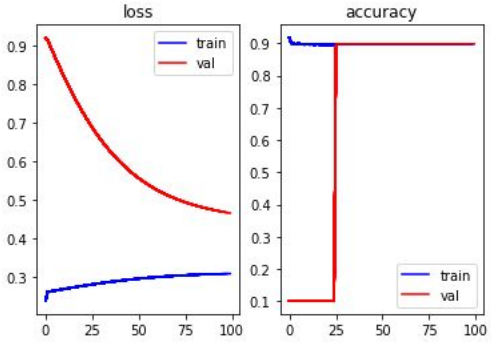
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  36  43  43  22   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  29  94
  190 242 253 252 212  28   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  22 187 252
  252 252 253 252 252 239 100   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 141 253 252 252
  252 252 253 252 252 252 252   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   9 176 246 253 182  51
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.14117648 0.16862746 0.16862746 0.08627451 0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.11372549 0.36862746
 0.74509805 0.9490196  0.99215686 0.9882353  0.83137256 0.10980392
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.08627451 0.73333335 0.9882353
 0.9882353  0.9882353  0.99215686 0.9882353  0.9882353  0.9372549
 0.39215687 0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.5529412  0.99215686 0.9882353  0.9882353
```

Normalización



Train accuracy: 0.99
Val accuracy: 0.70



Train accuracy: 0.89
Val accuracy: 0.89

Resultados distintas inicializaciones

Función	Tiempo	Accuracy	Loss
Xavier uniform	41.4	0.94	0.15
Valores fijados	52.5	0.95	0.14
Kaiming uniform	45.8	0.94	0.15
Kaiming normal	45.1	0.95	0.14
Orthogonal	46.9	0.94	0.16
Fan in and fan out	46.9	0.94	0.15
Random	47.3	0.94	0.18

Conclusiones

- Esta práctica nos ha ayudado a comprender cómo funcionan las redes neuronales.

- El trabajo en grupo ha facilitado la faena.

- Los aportes en el foro han sido de gran ayuda.

¡Gracias por vuestra atención!

Alejandro Garcia
Juan Plúa