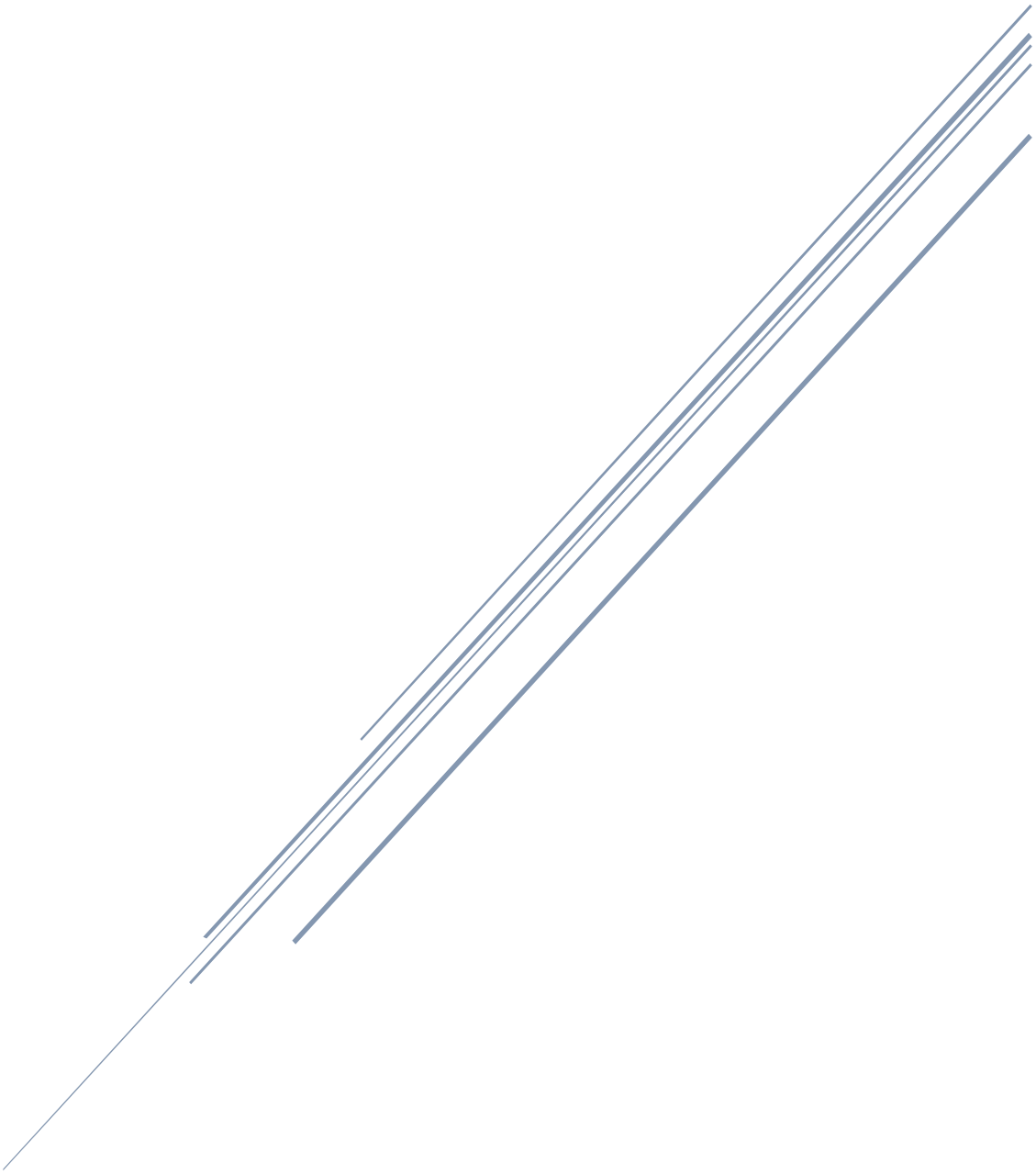
NEURAL NETWORKS

Module 4



v-cardona

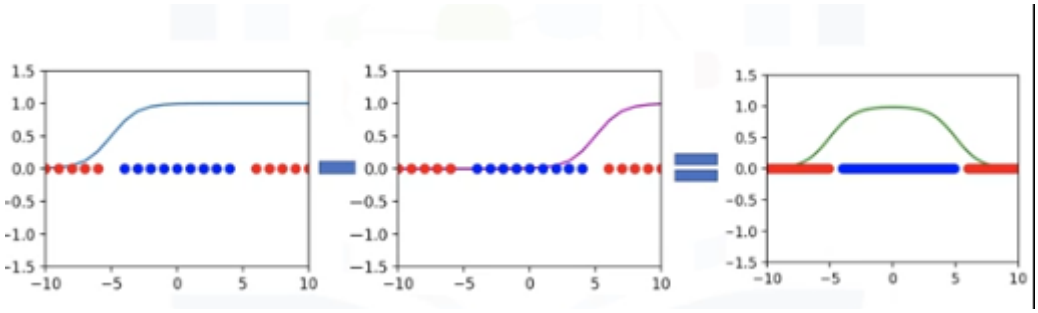
Deep Learning with Python and PyTorch

Table of contents

Neural networks 2
Neural networks with multiple dimensions 2
Multi-class networks 3
Neural networks for regression 3
Back-propagation 4
Vanishing gradient 4
Active functions 4
Tanh 4
Derivative 4
Relu 5
Building deep network in PyTorch 6
ModuleList 6

Neural networks

In a linear regression, we can separate classes with a line or with sigmoid functions, but if you cannot linear regression, it is not a good method. It that case we use neural networks where it is a combination of linear regression function to make only one



Essentially, we apply linear and sigmoid which is the artificial neuron, and then add different neuron. Hidden layer is the number of transformations that becomes inside de neural network.

Output layer transform the function to separate the points with a line.

- Defines class network:

```
class Net(nn.Module):
    # Constructor
    def __init__(self, D_in, H, D_out):
        super(Net, self).__init__()
        # hidden layer
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)
        # Define the first linear layer as an attribute, this is not good practice
        self.a1 = None
        self.l1 = None
    # Prediction
    def forward(self, x):
        self.l1 = self.linear1(x)
        self.a1 = torch.sigmoid(self.l1)
        yhat = torch.sigmoid(self.linear2(self.a1))
        return yhat
```

- It can be done with nn.Sequetial also(this has 2 dimension):

```
model= torch.nn.Sequential( torch.nn.Linear(2, 2),torch.nn.Sigmoid(), torch.nn.Linear(2,1), torch.nn.Sigmoid(),)
```

- First parameter is dimension, second is neurons and third output classes.

Neural networks with multiple dimensions

It seems like the more neurons you add, just like in 1-dimension, you can build more complicated functions. We do not have to change our neural network module or class, but, in the constructor, we have to specify the number of dimensions, first parameter.

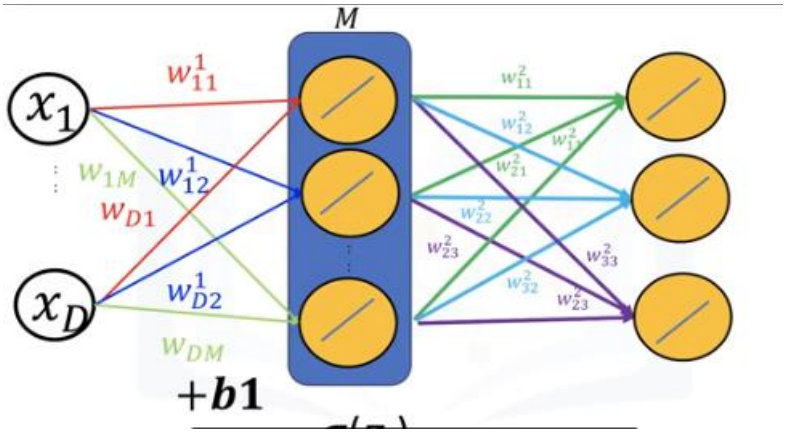
Overfitting is the problem of adding more neurons, because if we add more neurons it require more data and it can have problems with noise.

We can use validation data to determinate optimum number of hidden neurons and regularization.

Number of hidden layers are the arguments of the constructor of network minus 2 (minus input and ouput).

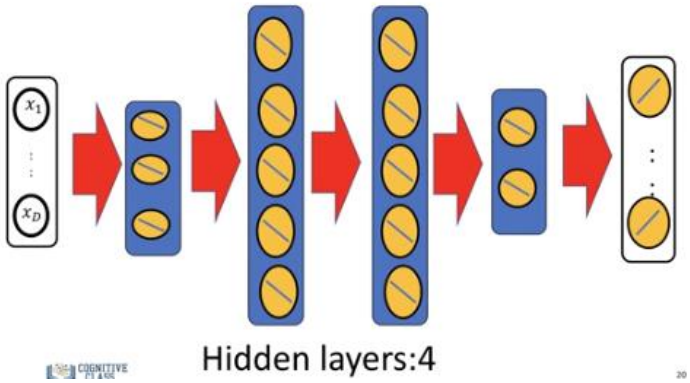
Multi-class networks

So multi-class neural networks function similar to regular neural networks. Instead of adding a logistic function in the output, we add a neuron for each class when a classify, and these neurons have their own set of parameters. Then to classify the actual class, which simply look at the output of each neuron and select the maximum value, exactly like Softmax function.



- First parameter, number of input dimensions.
- Second parameter, number of neurons.
- Third parameter of constructor of the network will be the number of classes, d_out.

Each hidden layer does not have to be the same size of neurons.



Neural networks for regression

For regression, the criterion or loss function that we should use is mean squared (MSELoss).

Back-propagation

Back propagation reduces a number of computations involved in calculating the derivative; and the problem of the sigmoid function i.e., the vanishing gradient.

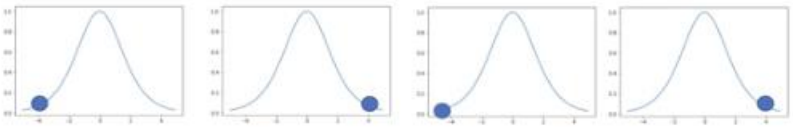
What back propagation does is it uses the derivative of the first parameter in the output layer to help us calculate the parameter of the next layer.

The reason for use backpropagation is because there are too many redundant computations and with these method they can be simplify.

Vanishing gradient

If the gradient of the first layer are very close to zero when we multiply per learning rate when we will calculate the upgrade values the next value will be similar to the original. So vanishing gradient is when we upgrade the equation but it does not because we are adding zero.

$$\frac{\partial l}{\partial w^1} \approx 0$$



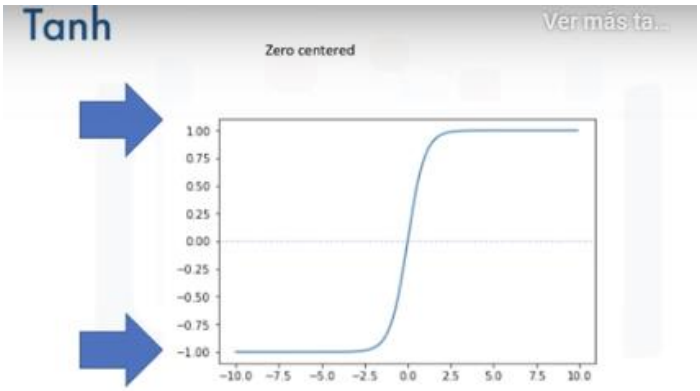
$$w^1 = w^1 + 0$$

Active functions

Tanh

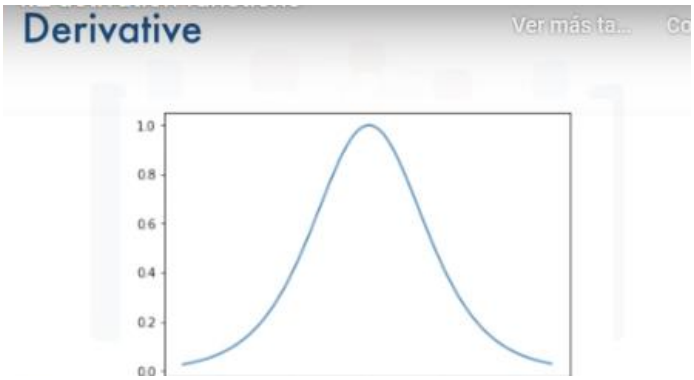
Tanh activation function, similar to sigmoid function, but goes from -1 to 1 and it is center in 0.

The problem is that the derivative is near zero in many regions, like sigmoid.



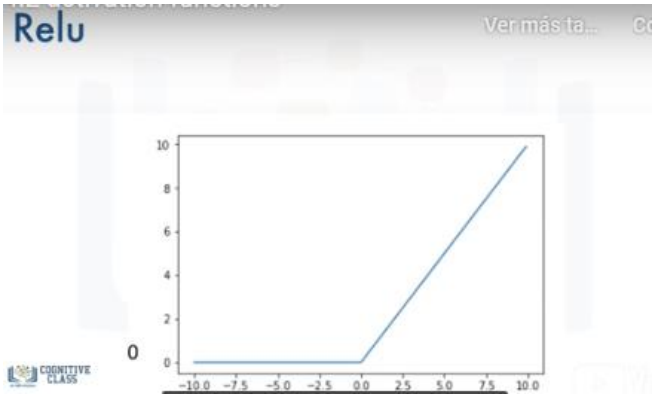
Derivative

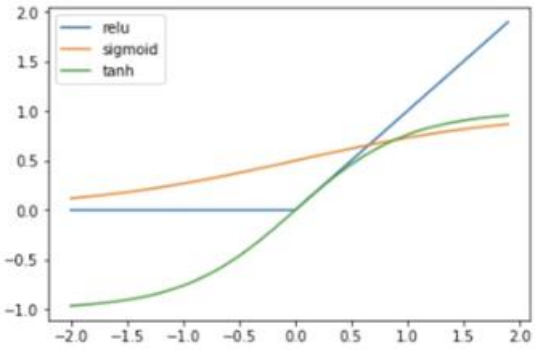
It has the problem that we comment before that if the values are closer to 0, there is no upgrade of the new value.



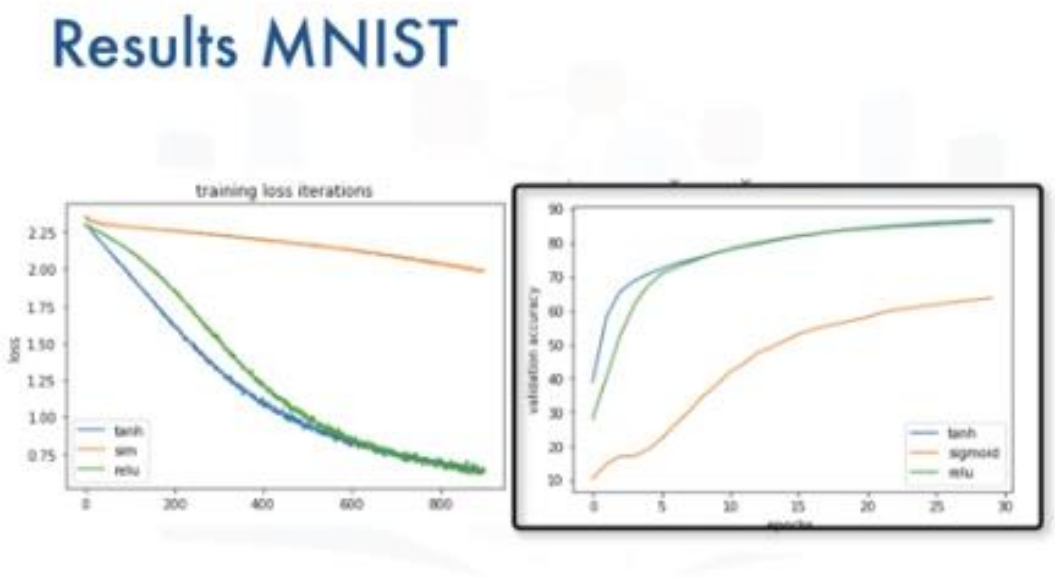
Relu

If the input is less than zero, the output is zero, and if the input is greater than zero, the output equals the input.





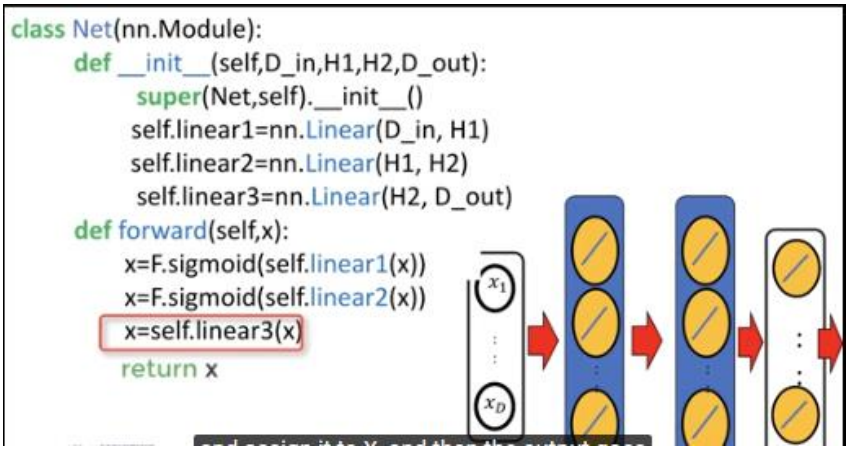
We usually apply this activation functions to the hidden layers.



Usually they suggest you should start with the Relu function because the Relu function is good for building neural deeper networks. So the more layers you have, the more likely that the Relu will perform better.

Building deep network in PyTorch

Number of hidden layers are the arguments of the constructor of network minus 2 (minus input and ouput). Remind that the value of the argument is the number of dimension and neurons.



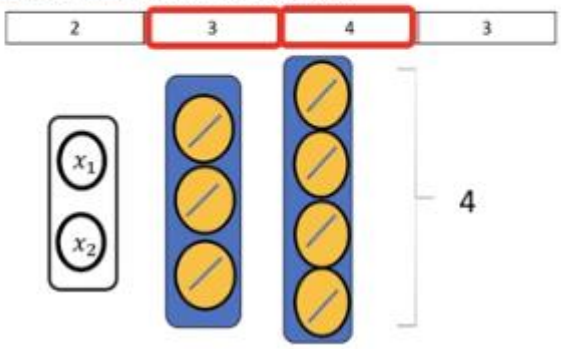
The Relu function is actually performing better than the Tanh function.

ModuleList

Instead of inputting all the layers manually, we will create a function that will create a variable number of layers with a variable number of neurons. We will create an input list and each element of the list consists of the number of neurons for a particular layer of a neural network.

```
class Net(nn.Module):
        def __init__(self,Layers):
                super(Net,self).__init__()
                self.hidden = nn.ModuleList()

                for input_size,output_size in zip(Layers,Layers[1:]):
                        self.hidden.append(nn.Linear(input_size,output_size))
```




```
Layers=[2,3,4,3]
model=Net(Layers)
```



```
def forward(self,activation):
    L=len(self.hidden)

    for (l,linear_transform)  in zip(range(L),self.hidden):

      if l<L-1:

        activation =F.relu(linear_transform (activation))

      else:
        activation =linear_transform (activation)

    return activation

Layers=[2,3,4,3]
model=Net(Layers)

yhat=model (x)
```