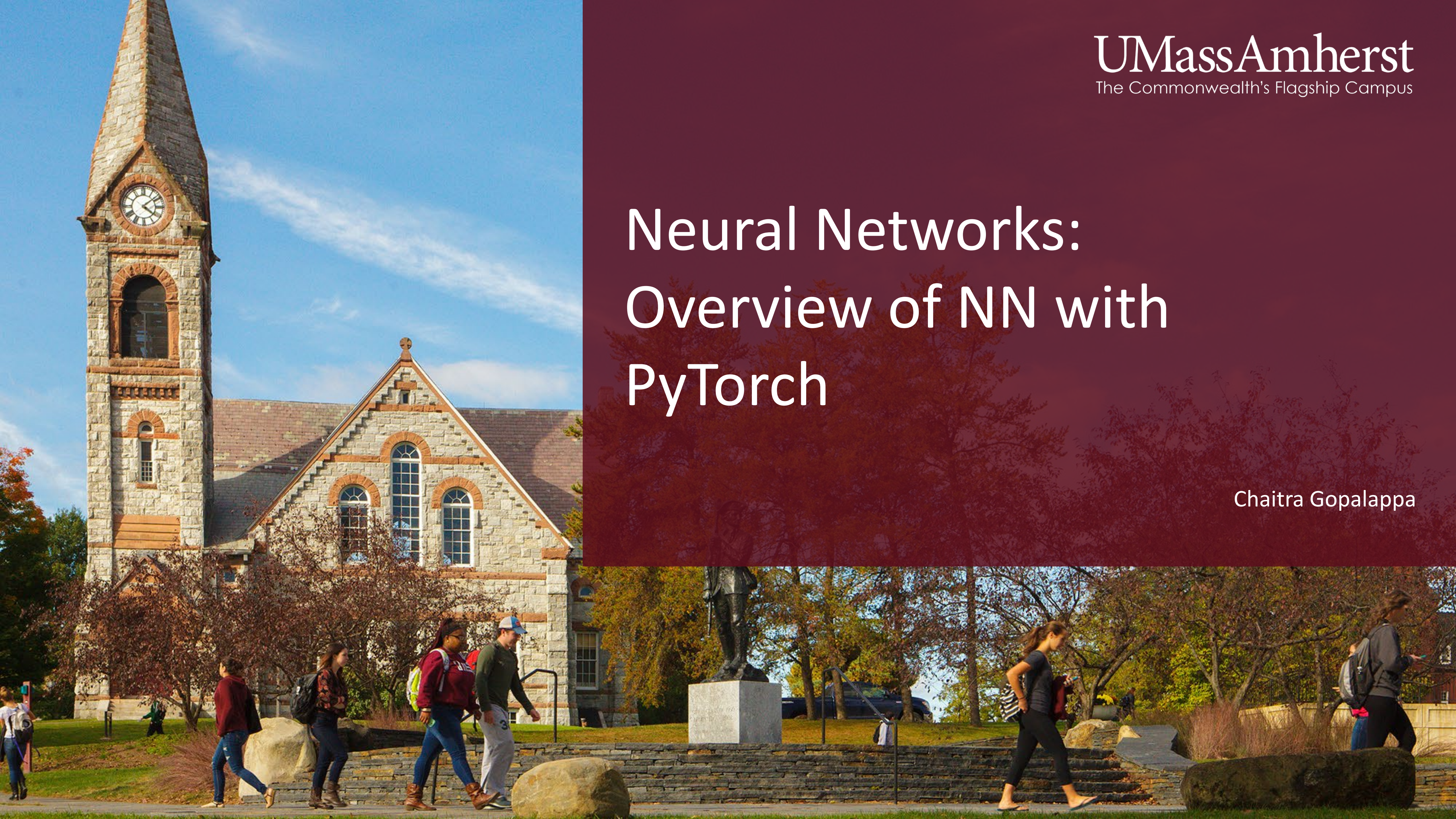
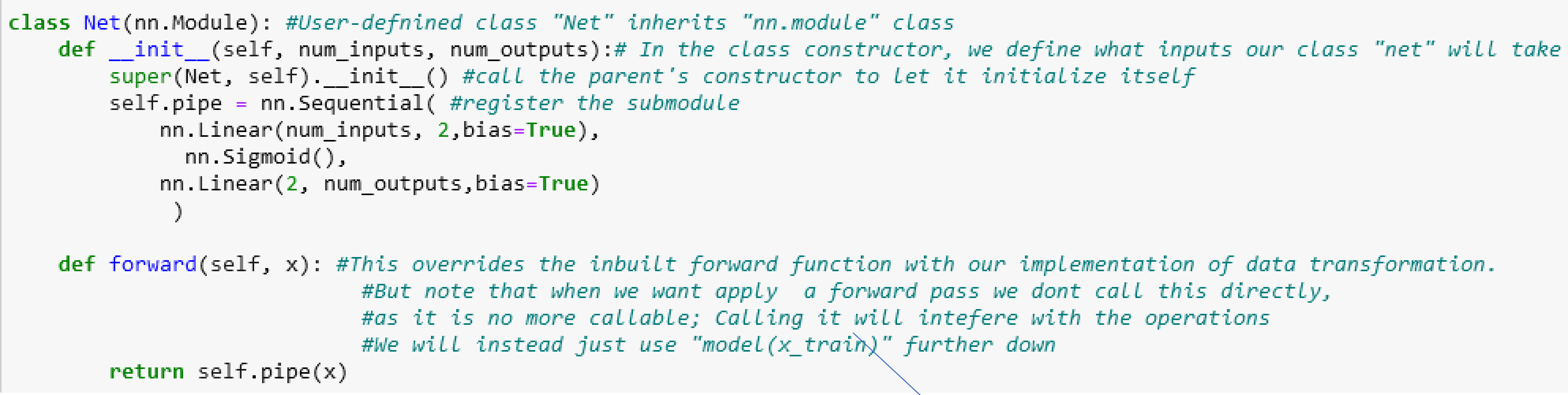


UMassAmherst
The Commonwealth's Flagship Campus

Neural Networks: Overview of NN with PyTorch

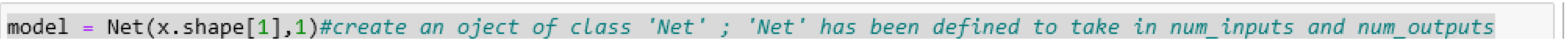
Chaitra Gopalappa

nn (neural network) module in pytorch



```
class Net(nn.Module): #User-defnined class "Net" inherits "nn.module" class
    def __init__(self, num_inputs, num_outputs):# In the class constructor, we define what inputs our class "net" will take
        super(Net, self).__init__() #call the parent's constructor to let it initialize itself
        self.pipe = nn.Sequential( #register the submodule
            nn.Linear(num_inputs, 2,bias=True),
              nn.Sigmoid(),
            nn.Linear(2, num_outputs,bias=True)
             )

    def forward(self, x): #This overrides the inbuilt forward function with our implementation of data transformation.
                            #But note that when we want apply  a forward pass we dont call this directly,
                            #as it is no more callable; Calling it will intefere with the operations
                            #We will instead just use "model(x_train)" further down
        return self.pipe(x)
```

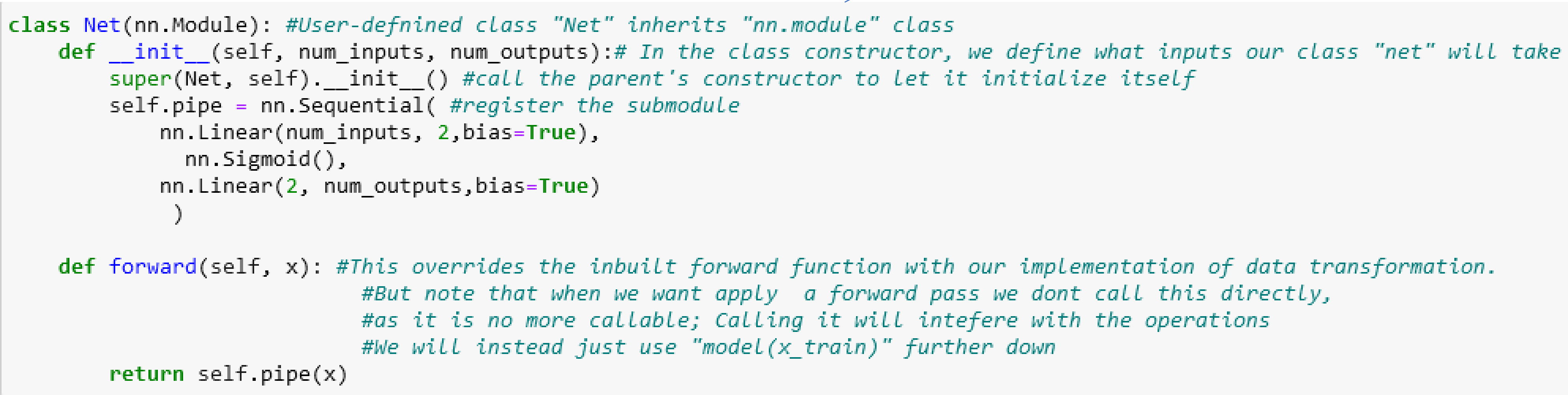
Typically, we can create methods that are callable from the object. We will see this when we go to RL.



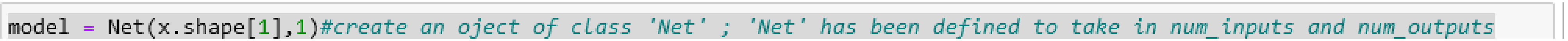
```
model = Net(x.shape[1],1)#create an oject of class 'Net' ; 'Net' has been defined to take in num_inputs and num_outputs
```

nn module in pytorch

Search nn.module in PyTorch , click source to look-up what it has.



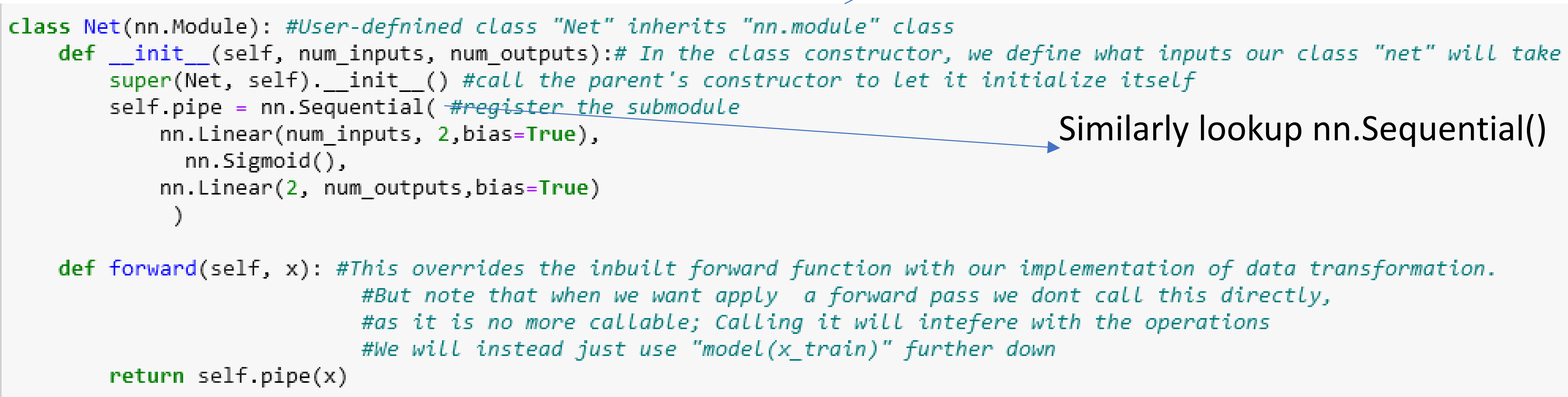
```
class Net(nn.Module): #User-defnined class "Net" inherits "nn.module" class
    def __init__(self, num_inputs, num_outputs):# In the class constructor, we define what inputs our class "net" will take
        super(Net, self).__init__() #call the parent's constructor to let it initialize itself
        self.pipe = nn.Sequential( #register the submodule
            nn.Linear(num_inputs, 2,bias=True),
              nn.Sigmoid(),
            nn.Linear(2, num_outputs,bias=True)
             )

    def forward(self, x): #This overrides the inbuilt forward function with our implementation of data transformation.
                            #But note that when we want apply  a forward pass we dont call this directly,
                            #as it is no more callable; Calling it will intefere with the operations
                            #We will instead just use "model(x_train)" further down
        return self.pipe(x)
```



```
model = Net(x.shape[1],1)#create an oject of class 'Net' ; 'Net' has been defined to take in num_inputs and num_outputs
```

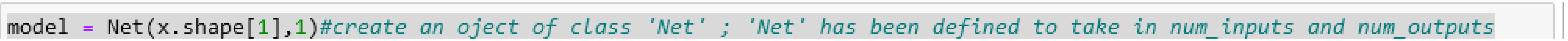
nn module in pytorch

Search nn.module in PyTorch , click source to look-up what it has.



```
class Net(nn.Module): #User-defnined class "Net" inherits "nn.module" class
    def __init__(self, num_inputs, num_outputs):# In the class constructor, we define what inputs our class "net" will take
        super(Net, self).__init__() #call the parent's constructor to let it initialize itself
        self.pipe = nn.Sequential( #register the submodule
            nn.Linear(num_inputs, 2,bias=True),
              nn.Sigmoid(),
            nn.Linear(2, num_outputs,bias=True)
             )

    def forward(self, x): #This overrides the inbuilt forward function with our implementation of data transformation.
                            #But note that when we want apply  a forward pass we dont call this directly,
                            #as it is no more callable; Calling it will intefere with the operations
                            #We will instead just use "model(x_train)" further down
        return self.pipe(x)
```

Similarly lookup nn.Sequential()

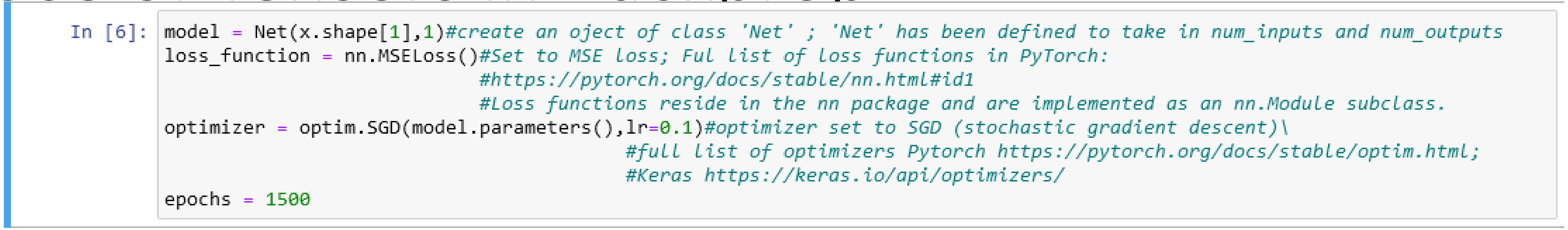


```
model = Net(x.shape[1],1)#create an oject of class 'Net' ; 'Net' has been defined to take in num_inputs and num_outputs
```

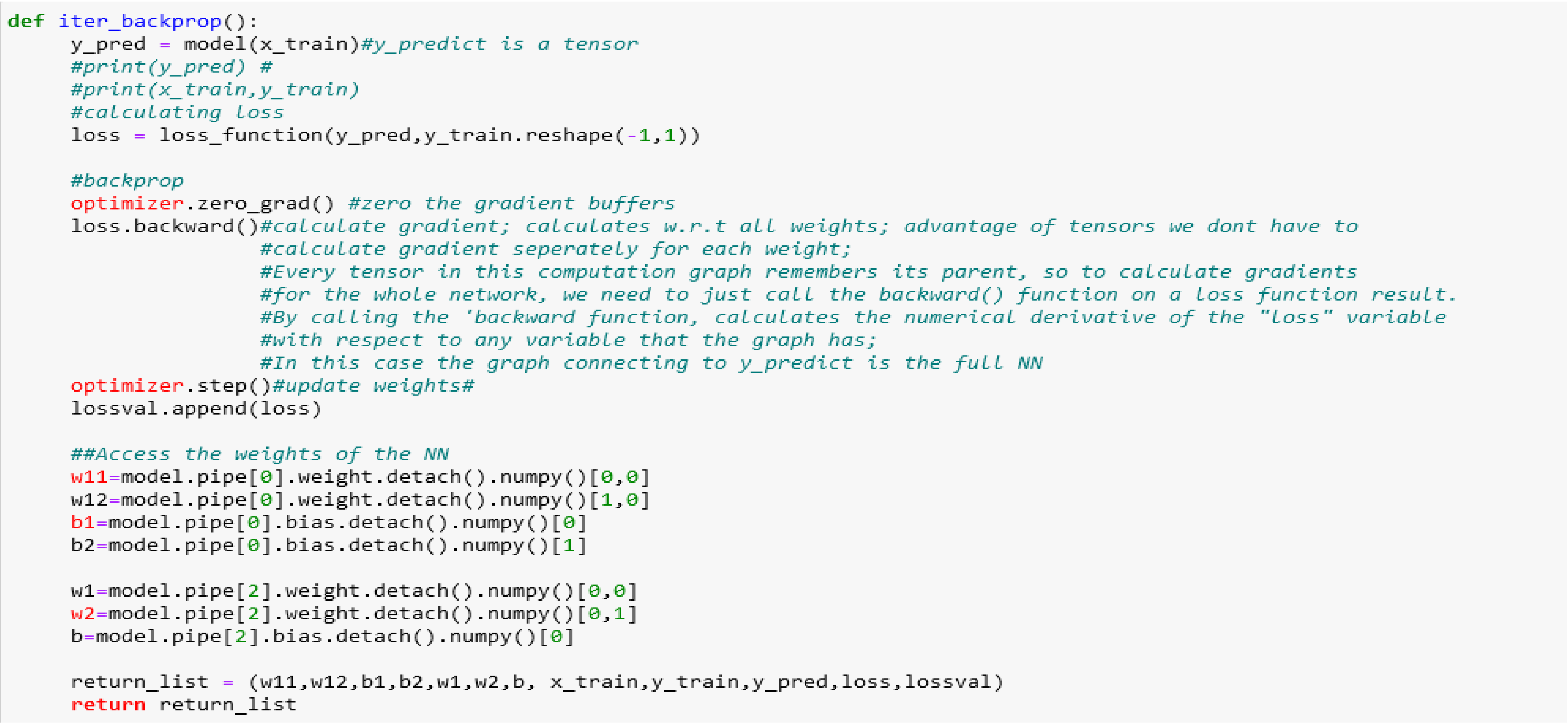
Tensors

- General: https://pytorch.org/tutorials/beginner/introyt/tensors_deeper_tutorial.html
- Tensors and autograd
    - https://pytorch.org/tutorials/beginner/examples_autograd/polynomial_autograd.html
- PyTorch: packages autograd into backprop so that we don’t have to use autograd

Use of Tensors in Backprop



```
In [6]: model = Net(x.shape[1],1)#create an oject of class 'Net' ; 'Net' has been defined to take in num_inputs and num_outputs
        loss_function = nn.MSELoss()#Set to MSE loss; Ful list of loss functions in PyTorch:
                                    #https://pytorch.org/docs/stable/nn.html#id1
                                    #Loss functions reside in the nn package and are implemented as an nn.Module subclass.
        optimizer = optim.SGD(model.parameters(),lr=0.1)#optimizer set to SGD (stochastic gradient descent)\
                                                 #full list of optimizers Pytorch https://pytorch.org/docs/stable/optim.html;
                                                 #Keras https://keras.io/api/optimizers/

        epochs = 1500
```



```
def iter_backprop():
    y_pred = model(x_train)#y_predict is a tensor
    #print(y_pred) #
    #print(x_train,y_train)
    #calculating loss
    loss = loss_function(y_pred,y_train.reshape(-1,1))

    #backprop
    optimizer.zero_grad() #zero the gradient buffers
    loss.backward()#calculate gradient; calculates w.r.t all weights; advantage of tensors we dont have to
                   #calculate gradient seperately for each weight;
                   #Every tensor in this computation graph remembers its parent, so to calculate gradients
                   #for the whole network, we need to just call the backward() function on a loss function result.
                   #By calling the 'backward function, calculates the numerical derivative of the "loss" variable
                   #with respect to any variable that the graph has;
                   #In this case the graph connecting to y_predict is the full NN
    optimizer.step()#update weights#
    lossval.append(loss)

    ##Access the weights of the NN
    w11=model.pipe[0].weight.detach().numpy()[0,0]
    w12=model.pipe[0].weight.detach().numpy()[1,0]
    b1=model.pipe[0].bias.detach().numpy()[0]
    b2=model.pipe[0].bias.detach().numpy()[1]

    w1=model.pipe[2].weight.detach().numpy()[0,0]
    w2=model.pipe[2].weight.detach().numpy()[0,1]
    b=model.pipe[2].bias.detach().numpy()[0]

    return_list = (w11,w12,b1,b2,w1,w2,b, x_train,y_train,y_pred,loss,lossval)
    return return_list
```

Varying NN architectures

```
self.pipe = nn.Sequential( #register the submodule
        nn.Linear(num_inputs, 10,bias=True),
          nn.ReLU(),
         nn.Linear(10, 10,bias=True),
         nn.ReLU(),
         nn.Linear(10, 10,bias=True),
         nn.ReLU(),
         nn.Linear(10, 10,bias=True),
         nn.Sigmoid(),
        nn.Linear(10, num_outputs,bias=True),
        nn.Tanh()
      )
```

Tensorboard

- Useful for monitoring lossfunction; to evaluate convergence

UMassAmherst

The Commonwealth's Flagship Campus