Tensors in one Dimension

- Convert a integer list with length 5 to a tensor:

  ints_to_tensor = torch.tensor([0, 1, 2, 3, 4])

  new_float_tensor = torch.FloatTensor([0, 1, 2, 3, 4])

  new_float_tensor.type()

  old_int_tensor = torch.tensor([0, 1, 2, 3, 4])

  new_float_tensor = old_int_tensor.type(torch.FloatTensor)

- The `tensor_obj.size()` helps you to find out the size of the `tensor_obj`. The `tensor_obj.ndimension()` shows the dimension of the tensor object

- The `tensor_obj.view(row, column)` is used for reshaping a tensor object.
  After you execute `new_float_tensor.view(5, 1)`, the size of `new_float_tensor` will be `torch.Size([5, 1])`.
  This means that the tensor object `new_float_tensor` has been reshaped from a one-dimensional tensor object with 5 elements to a two-dimensional tensor object with 5 rows and 1 column.
  Note: The number of elements in a tensor must remain constant after applying view.
  You get the same result as the previous example. The **-1** can represent any size. However, be careful because you can set only one argument as **-1**.

- Convert a numpy array to a tensor:

  numpy_array = np.array([0.0, 1.0, 2.0, 3.0, 4.0])

  new_tensor = torch.from_numpy(numpy_array)

- Convert a tensor to a numpy array:

  back_to_numpy = new_tensor.numpy()

- Calculate the mean for math_tensor and standard deviation:

  mean = math_tensor.mean()
  standard_deviation = math_tensor.std()

- `tensor_obj.max()` and `tensor_obj.min()`. These two methods are used for finding the maximum value and the minimum value in the tensor.

- *torch.sin(pi_tensor)*

- A useful function for plotting mathematical functions is `torch.linspace()`. `torch.linspace()` returns evenly spaced numbers over a specified interval. You specify the starting point of the sequence and the ending point of the sequence. The

parameter `steps` indicates the number of samples to generate. Now, you'll work with `steps = 5`.

- *plotVec([*
  {"vector": u.numpy(), "name": 'u', "color": 'r'},
  {"vector": v.numpy(), "name": 'v', "color": 'b'},
  {"vector": w.numpy(), "name": 'w', "color": 'g'}
  ])

- Calculate dot product of u, v:
  torch.dot(u,v)

Dimensional PyTorch Tensors

- Convert 2D List to 2D Tensor:

  twoD_list = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
  twoD_tensor = torch.tensor(twoD_list)

- *tensor_obj*`.ndimension()returns the dimensión of the tensor, row?`

- *tensor*.shape and tensor.size() returns the shape of the matrix in row and column. Both are the same. Size.[3,3]

- *tensor*.numbel() return the number of elements

- convert the Panda Dataframe to tensor:
  df = pd.DataFrame({'A':[11, 33, 22],'B':[3, 3, 2]})
  tensor = torch.tensor(df.values)

- Use *tensor_obj[row, column*] and *tensor_obj[row][column*] to access certain position.

- `torch.mm()` for calculating the multiplication between tensors. Producto de matrices normal. El Hadamard es a*b (elemento a elemento)

Derivatives

- requires_grad = True for take the derivate of the tensor
- function.backward() calculate the derivate respect x
- partial derivative : x.grad derivative at value set by requires grad
- Calculate the derivative with multiple values:
  x = torch.linspace(-10, 10, 10, requires_grad = True)
  *Y = x ** 2*
  *y = torch.sum(x ** 2)*
- Plot out the function and its derivative:
  y.backward()

```
plt.plot(x.detach().numpy(), Y.detach().numpy(), label = 'function')
plt.plot(x.detach().numpy(), x.grad.numpy(), label = 'derivative')
plt.xlabel('x')
plt.legend()
plt.show()
```

Simple dataset

- Define a dataset class with a getter (getitem) nad length (len) in the constructor you can pass a transform for the object.

```
class toy_set(Dataset):

    # Constructor with defult values
    def __init__(self, length = 100, transform = None):
        self.len = length
        self.x = 2 * torch.ones(length, 2)
        self.y = torch.ones(length, 1)
        self.transform = transform

    # Getter
    def __getitem__(self, index):
        sample = self.x[index], self.y[index]
        if self.transform:
            sample = self.transform(sample)
        return sample

    # Get Length
    def __len__(self):
        return self.len
```

- To apply a transform create a class with a constructor and executor method(call)
- Compose multiples transformations:

```
data_transform = transforms.Compose([add_mult(), mult()])
```