1. Common Practices

$.item()$ – get a single number out of a list/tensor as a python regular int/float

$set(list)$ – create a set object that contains unique values in the list

$sorted(list)$ – create a list object that contains sorted values in the list

1.1. Random Class

$random.choice(mylist)$ – choose a random sample from the list

$random.sample(mylist, k = n)$ – choose n random samples from the list

1.2. Dictionary

- Traversing through a dictionary
  - With keys

```
>>> likes = {"color": "blue", "fruit": "apple", "pet": "dog"}

>>> for key in likes:
...     print(key)
...
color
fruit
pet
```

  - With item

```
>>> for item in likes.items():
...     print(item)
...
('color', 'blue')
('fruit', 'apple')
('pet', 'dog')
```

1.3. Use Python to Copy files

```
# import module
import shutil

# copy the contents of the demo.py file to  a new file called demo1.py
shutil.copyfile('./demo.py', './demo1.py')
```

1.4. Try and Assert

- assert

```
#if condition returns False, AssertionError is raised:
assert x == "goodbye", "x should be 'hello'"
```

- try, except

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")

finally:
  print("The 'try except' is finished")
```

  - the finally block will be executed regardless
  - the else block will be executed if there’s no error

1.5. tqdm

$from\ tqdm.auto\ import\ tqdm$

$progress\_bar = tqdm(range(num\_steps))$

$progress\_bar.update(n)$ – n usually is 1, it is the amount of num_steps completed

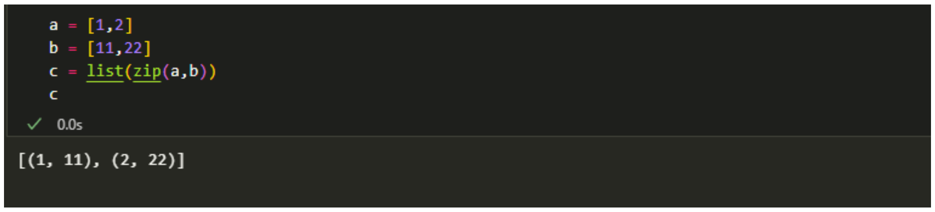
1.6. string functions

$'a\ d\ c\ b'.split(' ')$ → [‘a’, ‘d’, ‘c’, ‘b’]

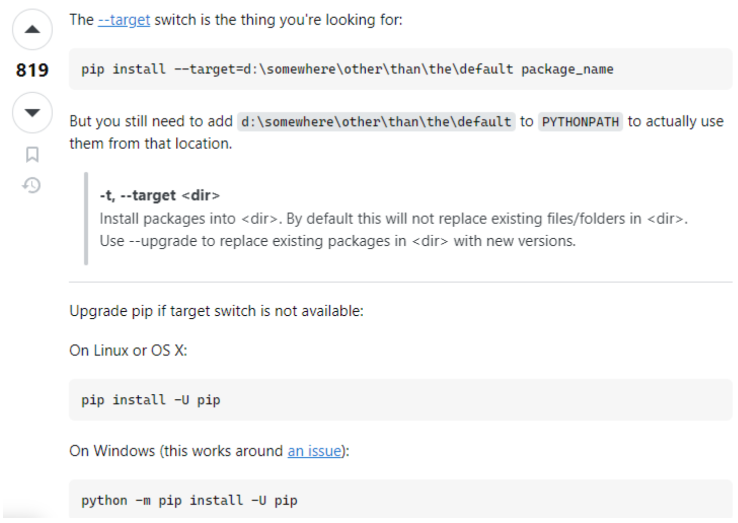
$'|'.join(['a','b','c'])$ → ‘a|b|c’

1.7. zip

- take two lists, iterate through each one and pair up the values from each list at the same index


```
a = [1,2]
b = [11,22]
c = list(zip(a,b))
c
```
✓ 0.0s
```
[(1, 11), (2, 22)]
```

1.8. pip install to a different location


819

The --target switch is the thing you're looking for:

```
pip install --target=d:\somewhere\other\than\the\default package_name
```

But you still need to add `d:\somewhere\other\than\the\default` to `PYTHONPATH` to actually use them from that location.

> **-t, --target <dir>**
> Install packages into <dir>. By default this will not replace existing files/folders in <dir>.
> Use --upgrade to replace existing packages in <dir> with new versions.

Upgrade pip if target switch is not available:

On Linux or OS X:

```
pip install -U pip
```

On Windows (this works around an issue):

```
python -m pip install -U pip
```

2. Python Classes

2.1. Basics

- Class definition

```
>>> class Dog:
...     species = "Canis familiaris"
...     def __init__(self, name, age):
...         self.name = name
...         self.age = age
...
```

- Print a class object without __str__ defined

```
>>> miles = Dog("Miles", 4)
```

```
>>> print(miles)
<__main__.Dog object at 0x00aeff70>
```

- Add __str__ definition

```
class Dog:
    # ...

    def __str__(self):
        return f"{self.name} is {self.age} years old"
```

```
>>> miles = Dog("Miles", 4)
>>> print(miles)
'Miles is 4 years old'
```

2.2. *Inheritance*

- You can inherit all attributes and methods from parent class, overwrite it, or extend it
    - Inherit – child's hair color is brown

```
# inheritance.py

class Parent:
    hair_color = "brown"

class Child(Parent):
    pass
```

- Overwrite the hair color to be purple

```
# inheritance.py

class Parent:
    hair_color = "brown"

class Child(Parent):
    hair_color = "purple"
```

- Extend

```python
# inheritance.py

class Parent:
    speaks = ["English"]

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.speaks.append("German")
```

2.3. *Example of inheritance – create Bread subclass of Dog class*

- Motivation: different breads of dog have different barking sound, but otherwise similar, so we need to create subclasses for bread with default barking sound, so we don't have to pass it as argument every time

- The parent class

```python
# dog.py

class Dog:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

- The child classes (initial)

```python
# dog.py

# ...

class JackRussellTerrier(Dog):
    pass

class Dachshund(Dog):
    pass

class Bulldog(Dog):
    pass
```

- Create instances

```
>>> miles = JackRussellTerrier("Miles", 4)
>>> buddy = Dachshund("Buddy", 9)
>>> jack = Bulldog("Jack", 3)
>>> jim = Bulldog("Jim", 5)
```

- Check which subclass it belongs to, or whether it is inherited

```
>>> type(miles)
<class '__main__.JackRussellTerrier'>
```

```
>>> isinstance(miles, Dog)
True
```

- Work on the bark sound

```
# dog.py

class Dog:
    # ...

    def speak(self, sound):
        return f"{self.name} barks: {sound}"

# ...
```

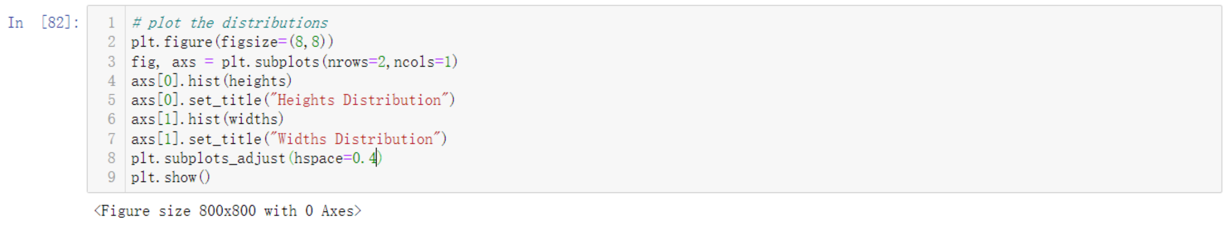
  - Access the parent class from inside a method of a child class by using super()

```
# dog.py

# ...

class JackRussellTerrier(Dog):
    def speak(self, sound="Arf"):
        return super().speak(sound)

# ...
```

```
>>> miles = JackRussellTerrier("Miles", 4)
>>> miles.speak()
'Miles barks: Arf'
```

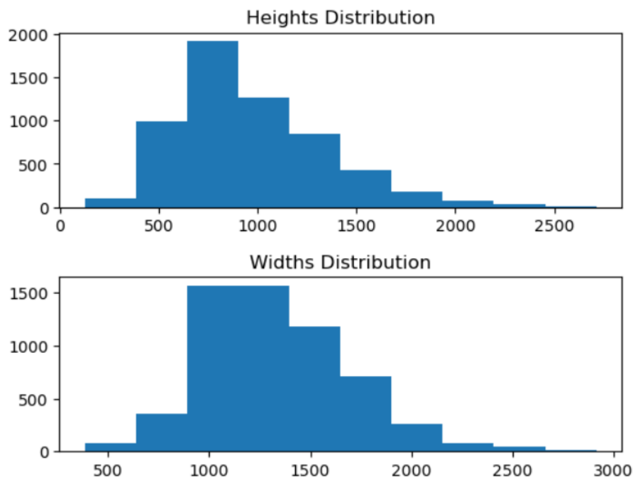
3. Plots

3.1. Create subplot



```
In [82]:
# plot the distributions
plt.figure(figsize=(8,8))
fig, axs = plt.subplots(nrows=2,ncols=1)
axs[0].hist(heights)
axs[0].set_title("Heights Distribution")
axs[1].hist(widths)
axs[1].set_title("Widths Distribution")
plt.subplots_adjust(hspace=0.4)
plt.show()
```

```
<Figure size 800x800 with 0 Axes>
```



- Subplot doc: https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html
- Adjust subplot spacing doc: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots_adjust.html

4. PyTorch

4.1. Device and check device

$$device = "cuda"\ if\ torch.cuda.is\_available(\quad)\ else\ "cpu"$$

$$next(model.parameters()).device$$

4.2. Common Practice for Train and Test Loop

$with\ torch.inference\_mode()$: set up inference mode for testing/validating

$pred\_labels = torch.round(model\_output)$ – get predicted labels for binary classification

$pred\_labels = model\_output.argmax(dim = 1)$ – get predicted labels for multiclass classification

$optimizer.zero\_grad()$ – clear the cache of optimizer (i.e. clear previous calculated gradients)

$loss.backward()$ - backpropagation

$optimizer.step()$ – update model parameters

$torch.eq(tensor1, tensor2).sum(\quad).item(\quad)$

4.3. Data Types

$a.long()$ – convert torch tensor a from float (usually) to int64

$a.flot()$ – convert torch tensor a from int (usually) to float32

4.4. Save & Load models

Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

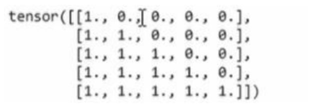
4.5. Extract Torch tensor element

$a[0].detach().cpu().numpy().item()$

4.6. Functions for LLM

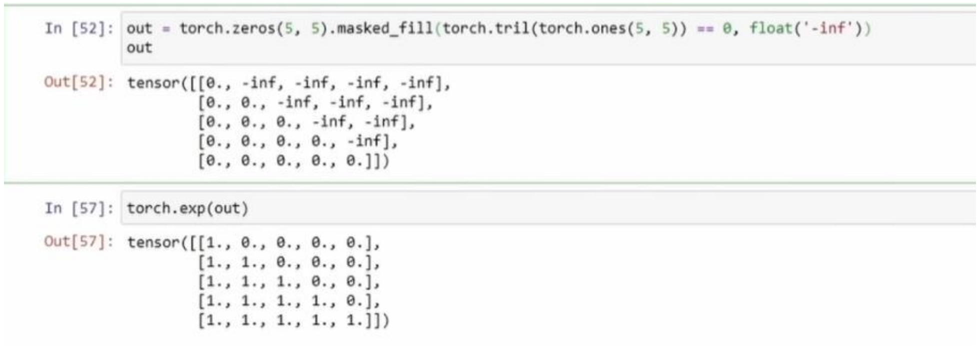
$torch.cat\big((a,b), dim = 0\big)$ – concatenate two tensors along the 0th dimension

$torch.tril\Big(\big(torch.ones(5,5)\big)\Big)$ – triangle lower

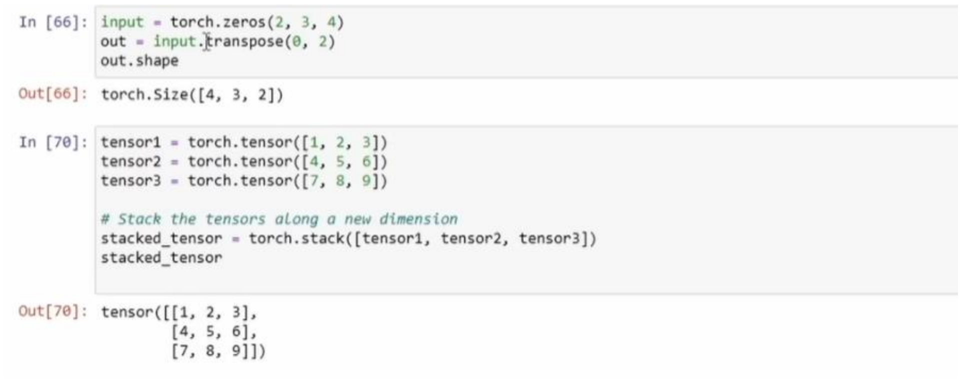


```
tensor([[1., 0., 0., 0., 0.],
        [1., 1., 0., 0., 0.],
        [1., 1., 1., 0., 0.],
        [1., 1., 1., 1., 0.],
        [1., 1., 1., 1., 1.]])
```

$torch.triu\Big(\big(torch.ones(5,5)\big)\Big)$ – triangle upper

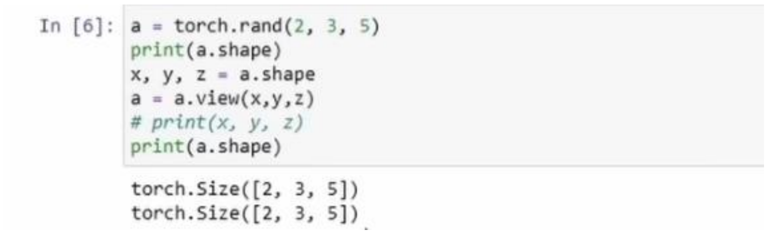


```
In [52]: out = torch.zeros(5, 5).masked_fill(torch.tril(torch.ones(5, 5)) == 0, float('-inf'))
         out

Out[52]: tensor([[0., -inf, -inf, -inf, -inf],
                 [0., 0., -inf, -inf, -inf],
                 [0., 0., 0., -inf, -inf],
                 [0., 0., 0., 0., -inf],
                 [0., 0., 0., 0., 0.]])

In [57]: torch.exp(out)

Out[57]: tensor([[1., 0., 0., 0., 0.],
                 [1., 1., 0., 0., 0.],
                 [1., 1., 1., 0., 0.],
                 [1., 1., 1., 1., 0.],
                 [1., 1., 1., 1., 1.]])
```



```
In [66]: input = torch.zeros(2, 3, 4)
         out = input.transpose(0, 2)
         out.shape

Out[66]: torch.Size([4, 3, 2])

In [70]: tensor1 = torch.tensor([1, 2, 3])
         tensor2 = torch.tensor([4, 5, 6])
         tensor3 = torch.tensor([7, 8, 9])

         # Stack the tensors along a new dimension
         stacked_tensor = torch.stack([tensor1, tensor2, tensor3])
         stacked_tensor

Out[70]: tensor([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])
```

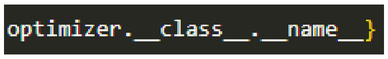
- View



```
In [6]: a = torch.rand(2, 3, 5)
        print(a.shape)
        x, y, z = a.shape
        a = a.view(x,y,z)
        # print(x, y, z)
        print(a.shape)

        torch.Size([2, 3, 5])
        torch.Size([2, 3, 5])
```

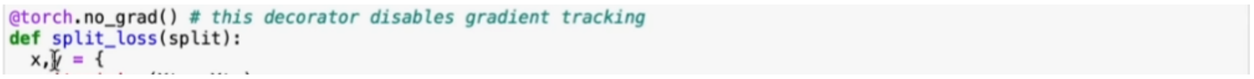
4.7. Optimizer

- Print out optimizer’s name



```
optimizer.__class__.__name__}
```

4.8. No_grad decorator



```
@torch.no_grad() # this decorator disables gradient tracking
def split_loss(split):
  x,y = {
```

- With the decorator, the subsequent function will not record the gradients

5. Work with files

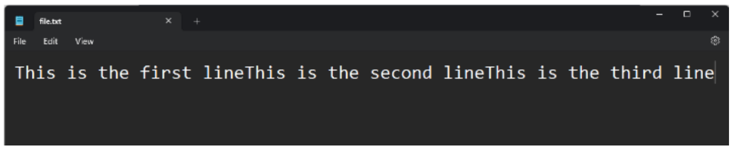
$file\_object \ = open(file\_name, mode)$

- mode:
  - "r" – read model – reading a file (default mode if not specified)
  - "w" – write mode – used for writing data to a file, create a new file if it doesn't exist, and overwrites the content if the file already exists
  - "a" – append mode – used for appending data to an existing file, create a new file if it doesn't exist
  - "x" – create mode – used for creating a new file, raises an error if the file exists
  - "b" – binary mode – used for reading and writing binary files, must be combined with other modes (e.g., "rb", "wb", …)

$file\_object.close()$ – close the file after performing all the operations

5.1. The write() method

```
f = open('file.txt', 'w')
f.write('This is the first line')
f.write('This is the second line')
f.write('This is the third line')

f.close()
```

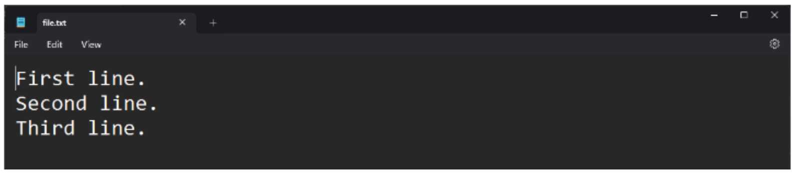


```
This is the first lineThis is the second lineThis is the third line
```

5.2. *The writelines() method*

2. Write the data using the **writelines()** method:`lines = ['First line.\n', 'Second line.\n', 'Third line.\n']`
`f.writelines(lines)`

3. Close the file using the **close()** method:`f.close()`



```
First line.
Second line.
Third line.
```

5.3. *Printing to a file*

1. Open the file using the **open()** function:`f = open('file.txt', 'w')`

2. Use the **print()** function with the **file** parameter set to the file object:`print('This is a sample text.', file=f)`

3. Close the file using the **close()** method:`f.close()`

5.4. *“with” statement*

- The "with" keyword automatically makes sure the file is properly closed once the block of code is executed

```
with open('example.txt', 'r') as file:
    content = file.read()
```

Link:

https://blog.enterprisedna.co/python-write-to-file/

6. OS module

6.1. File related

$os.path.exists()$ – checks whether a given path exists

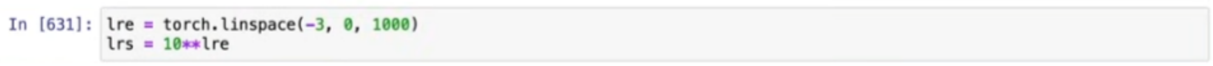
$os.makedirs()$ – creates a new directory or multiple directories (including intermediate ones) at the given path

$os.path.join()$ – joins various components of a path to form one absolute or relative file path (typically used to access files in subdirectories)

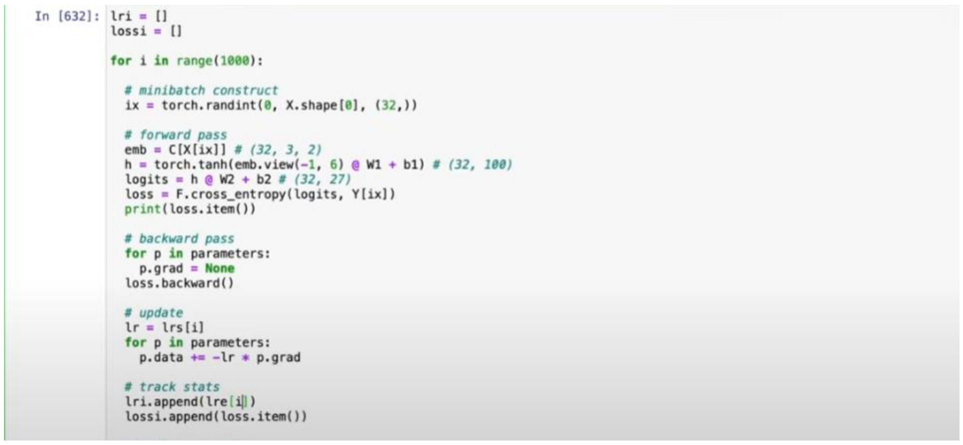
```
import os

# Check if directory exists, otherwise create it
dir_path = "example_dir"
if not os.path.exists(dir_path):
    os.makedirs(dir_path)

# Create file within the directory
file_path = os.path.join(dir_path, "test.txt")
with open(file_path, 'w') as file:
    file.write("This is an example.")
```

7. Deep Learning General

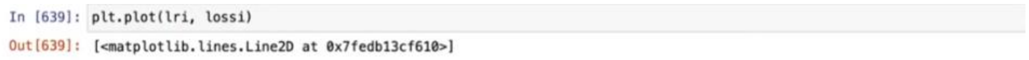
7.1. Finding good learning rate



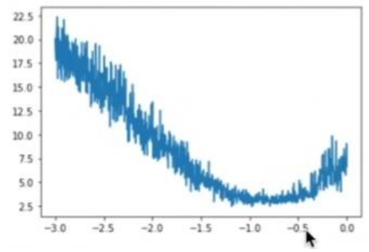
```
In [631]: lre = torch.linspace(-3, 0, 1000)
          lrs = 10**lre
```



```
In [632]: lri = []
          lossi = []

          for i in range(1000):

            # minibatch construct
            ix = torch.randint(0, X.shape[0], (32,))

            # forward pass
            emb = C[X[ix]] # (32, 3, 2)
            h = torch.tanh(emb.view(-1, 6) @ W1 + b1) # (32, 100)
            logits = h @ W2 + b2 # (32, 27)
            loss = F.cross_entropy(logits, Y[ix])
            print(loss.item())

            # backward pass
            for p in parameters:
              p.grad = None
            loss.backward()

            # update
            lr = lrs[i]
            for p in parameters:
              p.data += -lr * p.grad

            # track stats
            lri.append(lre[i])
            lossi.append(loss.item())
```



```
In [639]: plt.plot(lri, lossi)
Out[639]: [<matplotlib.lines.Line2D at 0x7fedb13cf610>]
```



- Set up learning rate from 1e-6 to 1e-0 with arbitrary space, train a batch (ideally with same parameters before training), record loss, and plot it, find optimal learning rate