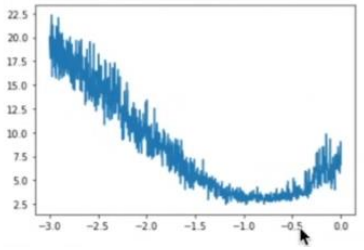
1. Deep Learning General

1.1. 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

1.2. Group two inputs words

- Context: embedding dimension = 10, input size = 8 words, batch size = 4, first layer hidden dimension = 200
- Original without grouping

```
In [246]: (torch.randn(4, 80) @ torch.randn(80, 200) + torch.randn(200)).shape
Out[246]: torch.Size([4, 200])
```

- Desired outcome

```
In [247]: (torch.randn(4, 4, 20) @ torch.randn(20, 200) + torch.randn(200)).shape
Out[247]: torch.Size([4, 200])
```

- In addition, we want to group adjacent words together

```
In [274]: e = torch.randn(4, 8, 10) # goal: want this to be (4, 4, 20) where consecutive 10-d vectors get concatenated
          torch.cat([e[:, ::2, :], e[:, 1::2, :]], dim=2).shape
Out[274]: torch.Size([4, 4, 20])
```

- Alternative, with *view*

```
In [275]: e = torch.randn(4, 8, 10) # goal: want this to be (4, 4, 20) where consecutive 10-d vectors get concatenated
          explicit = torch.cat([e[:, ::2, :], e[:, 1::2, :]], dim=2)
          explicit.shape
Out[275]: torch.Size([4, 4, 20])

In [277]: (e.view(4, 4, 20) == explicit).all()
Out[277]: tensor(True)
```

1.3. Use dictionary to encode decode tokens (character level)

```
[ ] # create a mapping from characters to integers
    stoi = { ch:i for i,ch in enumerate(chars) }
    itos = { i:ch for i,ch in enumerate(chars) }
    encode = lambda s: [stoi[c] for c in s] # encoder: take a string, output a list of integers
    decode = lambda l: ''.join([itos[i] for i in l]) # decoder: take a list of integers, output a string

    print(encode("hii there"))
    print(decode(encode("hii there")))

    [46, 47, 47, 1, 58, 46, 43, 56, 43]
    hii there
```

2. PyTorch

2.1. Device and check device

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

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

2.2. 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

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

2.3. Data Types

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

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

2.4. Save & Load models

Save:

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

Load:

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

2.5. Extract Torch tensor element

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

2.6. Torch common functions

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

$torch.tril\left((torch.ones(5,5))\right)$ – 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.]])
```

$torch.triu\left((torch.ones(5,5))\right)$ – 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])
```

- $torch.rand\_like()$ - Sample from N(0,1)
- $a.mul(b)$ – element wise multiplication

2.7. Optimizer

- Print out optimizer's name

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

2.8. 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

2.9. Transpose on more than 2 dimensional tensor

```
k = key(x)   # (B, T, 16)
q = query(x) # (B, T, 16)
wei =  q @ k.transpose(-2, -1) # (B, T, 16) @ (B, 16, T)
```

2.10. Use Sampler to customize DataLoader

- Ask Dataloader for only load a certain number of samples (number of batch will be number of samples / batch_size)

Initialize a random sampler providing `num_samples` and setting `replacement` to `True` *i.e.* the sampler is forced to draw instances multiple times if `len(ds) < num_samples`:

```
>>> sampler = RandomSampler(ds, replacement=True, num_samples=10)
```

```
val_sampler = torch.utils.data.RandomSampler(val_dataset,replacement=False,num_samples=100)
val_dataloader = DataLoader(dataset=val_dataset,batch_size=10,sampler=val_sampler)
```

21] ✓ 0.0s

```
for s_input, s_output in val_dataloader:
    print(f"input shape: {s_input.shape}")
    print(f"output shape: {s_output.shape}")
```
[22] ✓ 0.0s

```
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
input shape: torch.Size([10, 100])
output shape: torch.Size([10, 1])
```

```
len(val_dataloader)
```
[23] ✓ 0.0s

```
10
```

- Create training and validation split through sampler

In [4]:

```
import numpy as np
np.random.seed(100)
# construct validation dataset from training dataset
val_data_portion = 0.15
training_data_index = np.arange(len(train_dataset))
np.random.shuffle(training_data_index)
val_split = np.round(val_data_portion*len(training_data_index),0).astype(int)
val_idx = training_data_index[:val_split]
train_idx = training_data_index[val_split:]
print(f"first 10 indicies for train_idx: {train_idx[:10]}\n")

val_sampler = torch.utils.data.SubsetRandomSampler(val_idx)
train_sampler = torch.utils.data.SubsetRandomSampler(train_idx)

print(f"size of training data: {len(train_idx)}\nsize of validation data: {len(val_idx)}")
```

```
first 10 indicies for train_idx: [1480 3003 2004 2909 2480  598 3228 5157 1877  719]

size of training data: 4447
size of validation data: 785
```

In [5]:

```
from torch.utils.data import DataLoader
import os

BATCH_SIZE =32
workers = os.cpu_count()
train_dataloader = DataLoader(train_dataset,batch_size=BATCH_SIZE,drop_last=True,
                              sampler=train_sampler)
val_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE,drop_last=True,
                            sampler = val_sampler)
test_dataloader = DataLoader(test_dataset,batch_size=BATCH_SIZE,shuffle=False,drop_last=False)

print(f"There are {len(train_dataloader)} batches of training data, {len(val_dataloader)} batches of validation data"\
      + f" and {len(test_dataloader)} batches of test data")
```

```
There are 138 batches of training data, 24 batches of validation data and 20 batches of test data
```

2.11. Register buffer (non-trainable parameters)

$self.register\_buffer("var\ name", var)$

```python
# positional encoding from the paper
class PositionalEncoding(torch.nn.Module):
    def __init__(self, emb_dim, block_size,dropout_rate):
        # initialize PE matrix
        pe = torch.zeros((block_size,emb_dim))
        # assign values
        pe[:,0::2] = torch.sin(product_term)
        pe[:,1::2] = torch.cos(product_term)
        # create batch dimension
        pe = pe.unsqueeze(0)

        # make the parameters untrainable
        self.register_buffer("pos_enc", pe)
```

2.12. Print number of Parameters

```
print(round(sum(p.numel() for p in transformer.parameters())/1e6,2), 'M parameters')
```

[25] ✓ 0.0s

```
18.98 M parameters
```

2.13. Torch snippets for reporting

```
from torch_snippets import *
```

```
num_epochs = 5
log = Report(num_epochs)
```

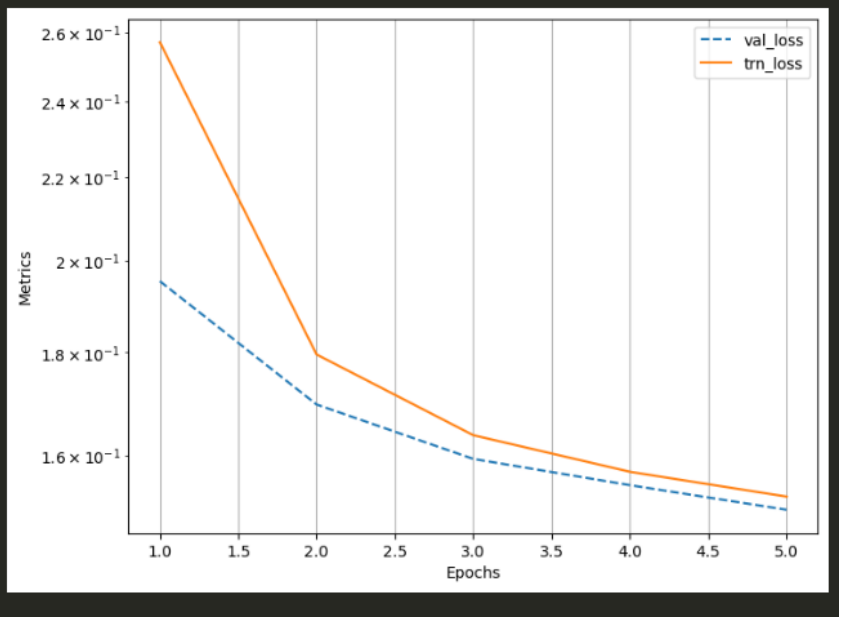
[14]

```
N_trn = len(trn_dl)
N_val = len(val_dl)
for epoch in range(num_epochs):
    for idx, (data, _) in enumerate(trn_dl):
        loss = train_batch(data,myAE,loss_fn,optimizer)
        log.record(pos=(epoch+(idx+1)/N_trn), trn_loss=loss, end="\r")
    for idx, (data, _) in enumerate(val_dl):
        loss = validate_batch(data, myAE, loss_fn)
        log.record(pos=(epoch+(idx+1)/N_val), val_loss = loss, end='\r')
    log.report_avgs(epoch+1)
```

[15]

```
EPOCH: 1.000  val_loss: 0.195  trn_loss: 0.257  (13.75s - 55.02s remaining)
EPOCH: 2.000  val_loss: 0.170  trn_loss: 0.180  (25.58s - 38.37s remaining)
EPOCH: 3.000  val_loss: 0.159  trn_loss: 0.164  (37.41s - 24.94s remaining)
EPOCH: 4.000  val_loss: 0.155  trn_loss: 0.157  (49.23s - 12.31s remaining)
EPOCH: 5.000  val_loss: 0.150  trn_loss: 0.153  (61.06s - 0.00s remaining)
```

```
[16]  1  log.plot_epochs(log=True)
                                                                    Pyth
...   100%|██████████| 106/106 [00:00<00:00, 5300.13it/s]
```



2.14. Use torchsummary for model summary

```
5  from torchsummary import summary
```

```
[4]   1  model = ConvAutoEncoder().to(device)
      2  summary(model, torch.zeros(2,1,28,28))
                                                                    Python
...   ==========================================================================================
      Layer (type:depth-idx)                   Output Shape              Param #
      ==========================================================================================
      ├─Sequential: 1-1                        [-1, 64, 3, 3]            --
      |    └─Conv2d: 2-1                       [-1, 32, 14, 14]          320
      |    └─ReLU: 2-2                         [-1, 32, 14, 14]          --
      |    └─MaxPool2d: 2-3                    [-1, 32, 7, 7]            --
      |    └─Conv2d: 2-4                       [-1, 64, 4, 4]            18,496
      |    └─ReLU: 2-5                         [-1, 64, 4, 4]            --
      |    └─MaxPool2d: 2-6                    [-1, 64, 3, 3]            --
      ├─Sequential: 1-2                        [-1, 1, 28, 28]           --
      |    └─ConvTranspose2d: 2-7              [-1, 32, 7, 7]            18,464
      |    └─ReLU: 2-8                         [-1, 32, 7, 7]            --
      |    └─ConvTranspose2d: 2-9              [-1, 16, 15, 15]          12,816
      |    └─ReLU: 2-10                        [-1, 16, 15, 15]          --
      |    └─ConvTranspose2d: 2-11             [-1, 1, 28, 28]           65
      |    └─Tanh: 2-12                        [-1, 1, 28, 28]           --
      ==========================================================================================
      Total params: 50,161
      Trainable params: 50,161
      Non-trainable params: 0
      Total mult-adds (M): 4.23
      ==========================================================================================
      Input size (MB): 0.01
      Forward/backward pass size (MB): 0.10
      Params size (MB): 0.19
      Estimated Total Size (MB): 0.30
      ------------------------------------------------------------------------------------------
```

2.15. nn.ModuleList vs. nn.Sequential

- nn.Sequential automatically feed output of previous layer to subsequent layer
- nn.ModuleList doesn't, we can define complex structure with this method

python Copy code

```python
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 1)
)
```

python Copy code

```python
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.layers = nn.ModuleList([
            nn.Linear(10, 20),
            nn.ReLU(),
            nn.Linear(20, 1)
        ])

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x
```

3. Tokenization

3.1. Byte Encoding

- Encode()

```
sample = text[:100]
sample
```
[4] ✓ 0.0s

```
'The Music of the Ainur There was Eru, the One, who in Arda is called lluvatar; and he made first the'
```

```
tokens = sample.encode("utf-8")
tokens
```
[39] ✓ 0.0s

```
b'The Music of the Ainur There was Eru, the One, who in Arda is called lluvatar; and he made first the'
```

- Convert byte-encoded text to numbers

```
tokens = sample.encode("utf-8")
print(int(tokens[0]))
```
[40] ✓ 0.0s

```
84
```

```
print(list(map(int,tokens))[:10])
```
[41] ✓ 0.0s

```
[84, 104, 101, 32, 77, 117, 115, 105, 99, 32]
```

- Alternatively

```
tokens = list(sample.encode("utf-8"))
# tokens = list(map(int,tokens))
tokens[:3]
```
[3] ✓ 0.0s

```
[84, 104, 101]
```

4. 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

4.1. *read binary vs. read*

103

You should use `"r"` for opening text files. Different operating systems have slightly different ways of storing text, and this will perform the correct translations so that you don't need to know about the idiosyncracies of the local operating system. For example, you will know that newlines will always appear as a simple `"\n"`, regardless of where the code runs.

You should use `"rb"` if you're opening non-text files, because in this case, the translations are not appropriate.

Share Improve this answer Follow

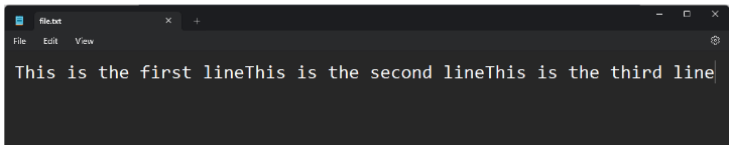
answered Feb 1, 2010 at 5:49

caf

236k • 40 • 326 • 466

4.2. *The write() method*

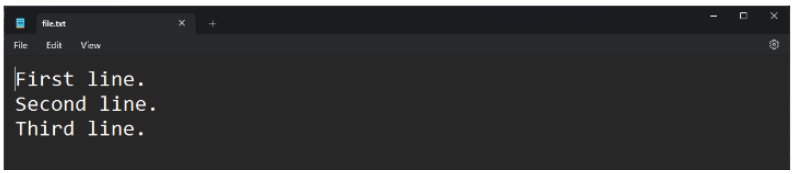
```
f = open('file.txt', 'w')
f.write('This is the first line')
f.write('This is the second line')
f.write('This is the third line')

f.close()
```



4.3. *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()`



4.4. 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()`

4.5. "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/

5. Work with Folders

5.1. Common functions involve os module

$os.path.exists()$ – 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.")
```

5.2. Delete folders

Use one of these methods:

- `pathlib.Path.unlink()` removes a file or symbolic link.
- `pathlib.Path.rmdir()` removes an empty directory.
- `shutil.rmtree()` deletes a directory and all its contents.

On Python 3.3 and below, you can use these methods instead of the `pathlib` ones:

- `os.remove()` removes a file.
- `os.unlink()` removes a symbolic link.
- `os.rmdir()` removes an empty directory.

6. Python Classes

6.1. 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'
```

6.2. 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")
```

6.3. 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'
```

6.4. Custom exception

```python
# Raise a custom exception
class CustomError(Exception):
    pass

raise CustomError("This is a custom error message")
```

7. 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

$itemgetter(*index\_list)(list)$ – retrieve multiple items from a list with a list of index, operator module

$string.find("pattern")$ – return the index of the substring

$list\big(map(function, target)\big)$ – apply the function on target

$mylist.index(element, start, end)$ – start and end are optional indices

$mystring.find(pattern, start, end)$ – find the index where the pattern occurred

7.1. Random Class

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

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

7.2. 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')
```

- Get multiple values with itemgetter

```
from operator import itemgetter

my_dict = {x: x**2 for x in range(10)}

itemgetter(1, 3, 2, 5)(my_dict)
#>>> (1, 9, 4, 25)
```

`itemgetter` will return a tuple if more than one argument is passed. To pass a list to `itemgetter`, use

```
itemgetter(*wanted_keys)(my_dict)
```

Keep in mind that `itemgetter` does not wrap its output in a tuple when only one key is requested, and does not support zero keys being requested.

- Save dictionary to pickle file

```
import pickle

with open('saved_dictionary.pkl', 'wb') as f:
    pickle.dump(dictionary, f)

with open('saved_dictionary.pkl', 'rb') as f:
    loaded_dict = pickle.load(f)
```

- Retrieve value from key

$dict.get(keyname, value)$ – value is optional, it is a value to return if the specified key does not exist

- Retrieve the value where the key is the largest with max

$top\_vale = \max(stats, key = stats.get)$ – stats is the dictionary

- Sort dictionary based on value

$sorted\_dict = sorted(dict.items(\ ), key = lambda\ x: x[1], reverse = True)$

- Get the key with the largest value in the dictionary

$top\_key = \max(dict, key = dict.get)$

7.3. 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')
```

7.4. 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

7.5. 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

7.6. string functions

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

$'|'.join(['a','b','c'])$ → 'a|b|c'

$string.strip()$ – removing leading and trailing spaces

7.7. zip

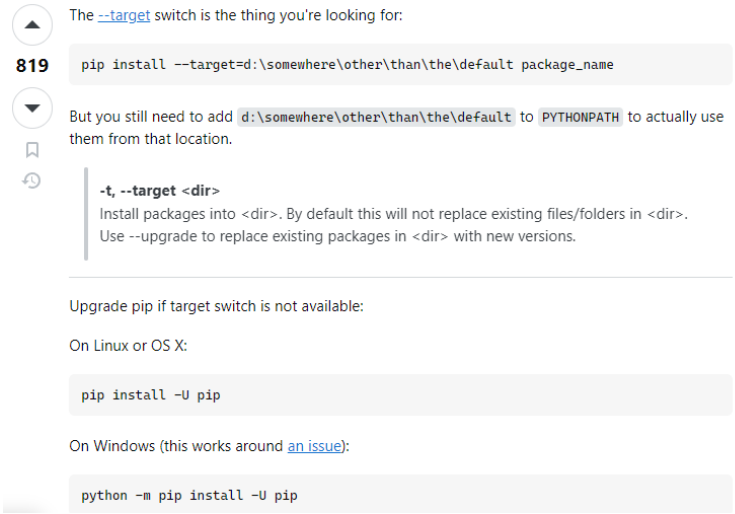
- take two lists, iterate through each one and pair up the values from each list at the same index

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

✓ 0.0s

```
[(1, 11), (2, 22)]
```

7.8. 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
```

7.9. enforcing argument data type

- force it to be string or None type

```
from typing import Optional

def fn(text: Optional[str]):
```

- for it to be string or integer

```python
from typing import Union

def fn(value: Union[str, int]):
```

7.10. saving datapoints with numpy

```
>>> with open('test.npy', 'wb') as f:
...     np.save(f, np.array([1, 2]))
...     np.save(f, np.array([1, 3]))
>>> with open('test.npy', 'rb') as f:
...     a = np.load(f)
...     b = np.load(f)
>>> print(a, b)
# [1 2] [1 3]
```

7.11. save array as text, load with np

```
[39]
array = np.load("training_data/train_85_split_vocabsize2k.npy")
len(array)
✓ 0.0s
895514
```

```
[51]
array2 = list(map(str,array))
array3 = array2
array3[:10]
✓ 0.1s
['32', '1446', '101', '1280', '84', '318', '760', '256', '1573', '358']
```

```
[72]
array4 = " ".join(array3)
array4[:20]
✓ 0.0s
'32 1446 101 1280 84 '
```

```
[73]
with open("training_data/train_85_split_vocabsize2k.txt", "w") as f:
    f.write(array4)
✓ 0.0s
```

```
[76]
b = np.loadtxt("training_data/train_85_split_vocabsize2k.txt",dtype='int')
b
✓ 0.1s
array([  32, 1446,  101, ...,  628,  387,   46])
```

```
[77]
(array == b).all()
✓ 0.0s
True
```

7.12. negating through Path

```
>>> from pathlib import Path

>>> p = Path('C:\\Program Files\\Internet Explorer\\iexplore.exe')

>>> str(p.parent)
'C:\\Program Files\\Internet Explorer'

>>> p.name
'iexplore.exe'

>>> p.suffix
'.exe'

>>> p.parts
('C:\\', 'Program Files', 'Internet Explorer', 'iexplore.exe')

>>> p.relative_to('C:\\Program Files')
WindowsPath('Internet Explorer/iexplore.exe')

>>> p.exists()
True
```

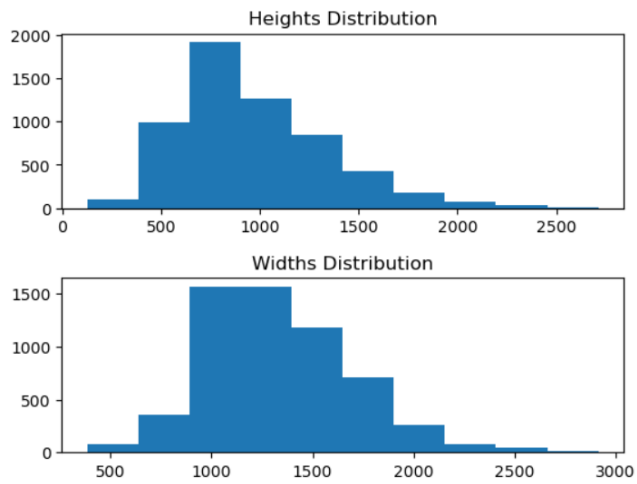
8. Plots

8.1. 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

8.2. Auto Encoder plot comparison

```python
for _ in range(4):
    idx = np.random.randint(len(val_ds))
    im, _ = val_ds[idx]
    im_pred = myAE(im[None])[0] # equivalent to im.unsqueeze(0)
    fig, ax = plt.subplots(1,2, figsize=(3,3))
    show(im[0],ax=ax[0], title="input")
    show(im_pred[0], ax=ax[1], title="prediction")
    plt.tight_layout()
    plt.show()
```

[18] Python