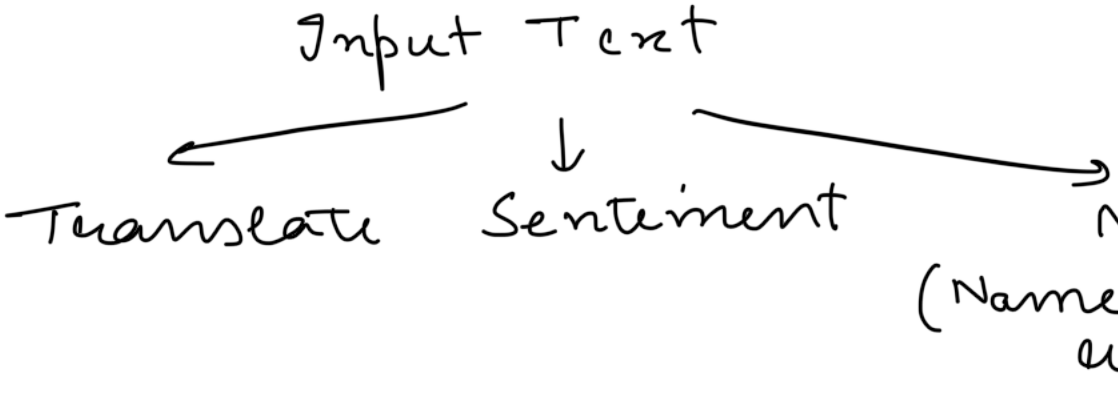
Week-1

L1 Intro to course & Recap of NN Models

- NLP enables computers to understand, interpret & gen. human lang.



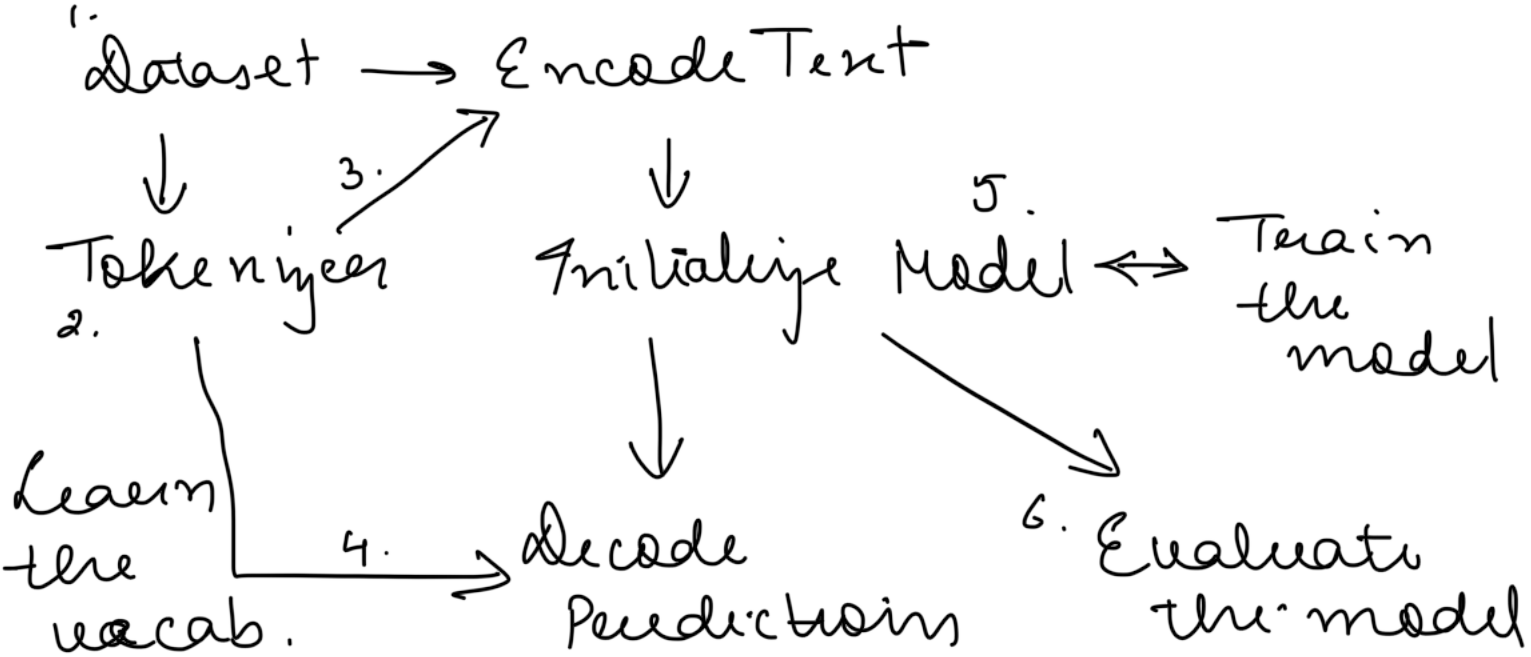
- n gram models → GPT 4, Llama (with emerging abilities)
- 4 types of architecture – MLP (backprop. algo. for learning the params. feed-forward fully connected neural network), CNN (CV, wt. sharing concept where the wts. in the kernels are shared across the input), RNN (recurrent LSTMs, transformers, etc. use the same wt. for all elements in the input seq).
- Now, it is easy to build these architectures using PyTorch & Tensorflow.
- Each layer can be derived from

`torch.nn.module` in PyTorch.

- MLP → The model is composed of linear layers & non-linear activations (assume ReLU) ∴, nn.linear and torch.relu.
- CNN → The model is composed of nn.linear, nn.Conv2d, MaxPool2D & torch.relu.
- RNN → The model is composed of nn.linear, nn.Embedding, nn.RNN. nn.RNN has nn.linear, torch.relu and other required modules.
- Transformer → Model is composed of nn.embedding, nn.Transformer, nn.Linear.

  nn.Transformer has nn.MultiHead Attention and other required modules.

L2 Why HF? Road Map



- HF does the heavy lifting
  1. Import datasets
  2. import tokenizers
  3. tokenizer.encode()
  4. tokenizer.decode()
  5. import transformers
  6. import evaluate
- HF has 150K datasets, 350K models, & 170K spaces.
- Modern NLP

  Prompt → Text (sentiment, summarize, fill in the blank, generate story)

  Lang. Modelling ↓

Output response conditioned on Prompt.

- wide range of NLP tasks — sentiment, machine transla", NER, question-answers, textual entailment, summary and generat".
- For each of these tasks we have 100s of datasets with 1000s of samples.
- Dataset can be varied sizes & formats understand the format & write a script to parse & load the text data.
- If memory is limited, we can stream the samples from the dataset.

L3 Datasets Module HF, loading

- access to datasets → diff. format, diff. size for various tasks.
  ↳ datasets module
- import datasets (movie reviews in SL)
  url — huggingface.co / datasets / —
- store it in cache memory. The cached data is stored in a memory-mapped columnar format, eg. less memory

- now load the dataset → data dict.
  train, test, unsupervised.
- each split has features & no. of rows

imbd_data['train']
↳ key

↳ now this is the train split
- remove unsupervised (using pop on dict)

dataset.pop('unsupervised')
- load("____", split="train")
  ↳ to download only train split.
- we can train on certain dataset, while show the evaluⁿ on some other dataset
- train → can also be split

train.train_test_split(test_size=0.2)
- data
  - train.csv
  - test.csv

  } same no. of columns and same col. name

data_files = [_, _]

load_dataset("csv", data_files = ____)
- convert dataset from any format to pyarrow format by saving it to the disk.
- load it back from load_from_disk funcⁿ.

L4 Accessing Samples

- index = 1000
  ex = dataset['train'][1000] } or send a list
  pprint(ex)

  label — } dictionary obj.
  text —

  [ ].select([idx])

- Now explore translation dataset
  wmt | wmt14

  get_dataset_config_name("___")
  → cs-en, ___ -

  get_dataset_split_names("___", "hi-en")
  → train, validation, test

- if you want to get the entire dataset, we can combine the splits
- features → define the int. structure.
  It specifies the underlying serialized task.

  pprint(dataset['train'].features)

- Glue: data → sub-config: map

L5 Common Methods

- datasetdict → filter, map, concatenate
  ↳ operates on all samples across the splits

- filtering → no. of words in a sentence should be ≥ 100.

```
nu = 1000
data.filter(lambda eg: len(eg['text']
    .split(' ')) > nu)
```

- map → add prefix IMDB: to all the samples.

```
eg['text'] = "IMDB: " + eg['text']
dataset.map(add_prefix)
```

- concatenate → same features & same no. of splits
  ↳ eg. IMDB + rotten tomatoes

```
datasets.concatenate_datasets
    ([ __ , __ ], axis = 0)
```

- interleaving datasets → n-skewed datasets, that is no. of samples in the dataset might differ drastically

```
interleave_datasets([ __ , __ ],
    probabs. = [0.6, 0.4])
stopping_strategy = first_exhausted
  ↳ default
```

- iterable dataset → use on the fly dataset.

```
load( __ __ __ , streaming = True)
```

the other methods like map, filter
also does its work on the fly