CLASSIFYING NAMES WITH A CHARACTER-LEVEL RNN

Prepared by:
Nesma Mahmoud

For this homework I created a **3 files.**
First file for reading the dataset.

The dataset that I used is the names files, I read all the names and the categories and put them into a TextToInstance field as explained in the practice session.

This Instance has 2 things inside, One of them is the language name and the other is the names of people for this language

The Dataset reader file:

```
@DatasetReader.register('data-reader')
class NameDatasetReader(DatasetReader):
    def __init__(self,tokenizer = None ,token_indexers: Dict[str, TokenIndexer] = None) -> None:
        super().__init__(lazy=False)
        self.tokenizer = tokenizer or CharacterTokenizer()

        self.token_indexers = token_indexers or {"tokens": SingleIdTokenIndexer()}

    def text_to_instance(self, tokens: List[Token], tags: List[str] = None) -> Instance:
        name_field = TextField(tokens, self.token_indexers)
        fields = {"name": name_field}

        if tags:
#             label_field = LabelField(tags)
            label_field = SequenceLabelField(labels=[tags]*len(tokens), sequence_field=name_field)
            fields["label"] = label_field

        return Instance(fields)

    def findFiles(self,path): return glob.glob(path)


    all_letters = string.ascii_letters + " .,;'"
    n_letters = len(all_letters)

    def unicodeToAscii(self,s):
     return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in self.all_letters
     )
    category_lines = {}
    all_categories = []

    # Read a file and split into lines
    def readLines(self,filename):
      lines = open(filename, encoding='utf-8').read().strip().split('\n')
      return [self.unicodeToAscii(line) for line in lines]

    def _read(self, file_path: str)-> Iterator[Instance]:
      for filename in self.findFiles(file_path):
#             print(filename)

            category = os.path.splitext(os.path.basename(filename))[0]
#             print (category)
            self.all_categories.append(category)
            lines = self.readLines(filename)
            for line in lines:
              yield self.text_to_instance([Token(word) for word in line], category)
            self.category_lines[category] = lines
```

Model File:
After Reading the dataset and preparing the input to the model, I created the model file itself.

The model consists of a word embedding layer, and encoder layer and a linear layer.

I used the Categorical Accuracy and the sequence cross entropy as the loss function in the model.

```
@Model.register('name-reader')
class NamesTagger(Model):
    def __init__(self,
                 word_embeddings: TextFieldEmbedder,
                 encoder: Seq2SeqEncoder,
                 vocab: Vocabulary) -> None:
        super().__init__(vocab)
        self.word_embeddings = word_embeddings
        self.encoder = encoder
        self.hidden2tag = torch.nn.Linear(in_features=encoder.get_output_dim(),
                                          out_features=vocab.get_vocab_size('labels'))
        self.accuracy = CategoricalAccuracy()
        self.loss = torch.nn.CrossEntropyLoss()
#         self.loss = torch.nn.BCEWithLogitsLoss()
#     def forward(self,sentence,labels) -> Dict[str, torch.Tensor]:

    def forward(self,
                name: Dict[str, torch.Tensor],
                label: torch.Tensor = None) -> torch.Tensor:

        mask = get_text_field_mask(name)
#         print("MY names",name)
#         print("Mask",mask)
        embeddings = self.word_embeddings(name)
        encoder_out = self.encoder(embeddings, mask)
#         print(encoder_out.shape)
        logits = self.hidden2tag(encoder_out)


        output = {"logits": logits}
#         print("output",output)
#         print("labels",label)
#         print(label.shape)
#         print(logits.shape)
        if label is not None:
            self.accuracy(logits, label,mask)
            output["loss"] = sequence_cross_entropy_with_logits(logits, label, mask)
#             output["loss"] = self.loss(logits,label)
        return output


    def get_metrics(self, reset: bool = False) -> Dict[str, float]:
        return {"accuracy": self.accuracy.get_metric(reset)}
```

The model reached accuracy: 0.5718, loss: 1.5425.
and this is how it started running

For Testing purposes I used spyder first for running the code, to ensure that every thing is going good, then I moved to use the configuration files.

```
In [32]: runfile('/home/nesma/SemesterII/NLP/hw4/data/NameTagger.py',
wdir='/home/nesma/SemesterII/NLP/hw4/data')


0it [00:00, ?it/s]
accuracy: 0.5287, loss: 1.8533 ||:  36%|███▌      | 3668/10072
[09:23<16:23,  6.51it/s]
4384it [00:00, 7356.13it/s]
8169it [00:00, 9700.67it/s]
15011it [00:00, 13064.26it/s]
20143it [00:00, 23060.88it/s]
0it [00:00, ?it/s]
4it [00:00, 4489.49it/s]
  0%|          | 0/20147 [00:00<?, ?it/s]
 95%|█████████▍| 19219/20147 [00:00<00:00, 192183.77it/s]
100%|██████████| 20147/20147 [00:00<00:00, 189934.13it/s]
  0%|          | 0/10072 [00:00<?, ?it/s]
accuracy: 0.0000, loss: 3.0970 ||:   0%|          | 1/10072
[00:00<2:46:30,  1.01it/s]
accuracy: 0.1640, loss: 2.9047 ||:   0%|          | 23/10072
[00:01<1:56:31,  1.44it/s]
accuracy: 0.3719, loss: 2.6433 ||:   0%|          | 47/10072
[00:01<1:21:35,  2.05it/s]
accuracy: 0.3619, loss: 2.5394 ||:   1%|          | 78/10072 [00:01<57:05,
2.92it/s]
accuracy: 0.4169, loss: 2.3687 ||:   1%|          | 112/10072
[00:01<39:58,  4.15it/s]
accuracy: 0.4414, loss: 2.2558 ||:   1%|          | 146/10072
[00:01<28:02,  5.90it/s]
accuracy: 0.4760, loss: 2.1701 ||:   2%|          | 179/10072
[00:01<19:42,  8.36it/s]
accuracy: 0.4879, loss: 2.1240 ||:   2%|          | 211/10072
[00:01<13:54, 11.81it/s]
accuracy: 0.4979, loss: 2.0904 ||:   2%|          | 243/10072
[00:01<09:51, 16.61it/s]
accuracy: 0.4944, loss: 2.0656 ||:   3%|          | 275/10072
[00:01<07:02, 23.20it/s]
accuracy: 0.5047, loss: 2.0280 ||:   3%|          | 308/10072
[00:02<05:03, 32.15it/s]
accuracy: 0.5174, loss: 1.9855 ||:   3%|          | 339/10072
[00:02<03:41, 43.89it/s]
accuracy: 0.5172, loss: 1.9731 ||:   4%|          | 374/10072
[00:02<02:43, 59.48it/s]
accuracy: 0.5111, loss: 1.9862 ||:   4%|          | 409/10072
[00:02<02:02, 79.15it/s]
accuracy: 0.5125, loss: 1.9925 ||:   4%|          | 442/10072
[00:02<01:34, 102.31it/s]
accuracy: 0.5033, loss: 2.0094 ||:   5%|          | 476/10072
[00:02<01:14, 129.29it/s]
accuracy: 0.5038, loss: 2.0113 ||:   5%|          | 509/10072
[00:02<01:00, 156.93it/s]
accuracy: 0.5029, loss: 2.0199 ||:   5%|          | 542/10072
```

The Third file is the Predictor file:

```
from overrides import overrides

from allennlp.common.util import JsonDict
from allennlp.data import Instance
from allennlp.predictors.predictor import Predictor


@Predictor.register('name-Predictor')
class Names_Predictor(Predictor):

    @overrides
    def predict_json(self, json_dict: JsonDict) -> JsonDict:
        name = json_dict['name']
        instance = self._dataset_reader.text_to_instance(name=name)

        label_dict = self._model.vocab.get_index_to_token_vocabulary('label')
        all_labels = [label_dict[i] for i in range(len(label_dict))]

        return {"instance": self.predict_instance(instance), "all_labels": all_labels}

    @overrides
    def _json_to_instance(self, json_dict: JsonDict) -> Instance:
        name = json_dict['name']
        return self._dataset_reader.text_to_instance(name=name)
```

for the configuration file, I set some local variables as the patience, hidden dim, learning rate, etc. Please find it bellow

```
local char_embedding_dim = 128;
local hidden_dim = 128;
local num_epochs = 150;
local patience = 10;
local batch_size = 8;
local learning_rate = 0.1;

{
    "train_data_path": './data/train.txt',
    "validation_data_path": './data/val.txt',
    "dataset_reader": {
        "type": "names-tagger-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "names-tagger",
        "word_embeddings": {

            "token_embedders": {
                "token_characters": {
                    "type": "character_encoding",
                    "embedding": {
                        "embedding_dim": char_embedding_dim,
                    },
                    "encoder": {
                        "type": "lstm",
                        "input_size": char_embedding_dim,
                        "hidden_size": char_embedding_dim
                    }
                }
            }
        },
        "encoder": {
            "type": "lstm",
            "input_size": embedding_dim,
            "hidden_size": hidden_dim
        }
    },
    "iterator": {
        "type": "basic",
        "batch_size": batch_size,
    },
    "trainer": {
        "num_epochs": num_epochs,
        "optimizer": {

            "type": "adam"
        },
        "patience": patience
    }
}
```

Please find the code in the github repository:

https://github.com/nesmaAlmoazamy/Names-Classification-Using-AllenNLP