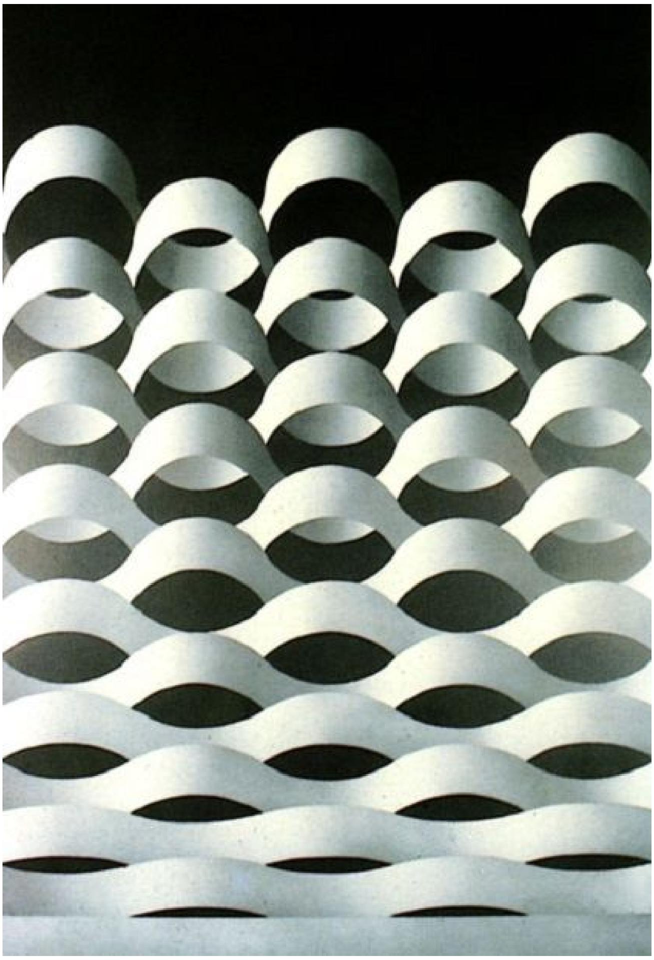
AllenNLP Workshop Exercise Booklet



Luke D. Gessler
lukegessler@gmail.com

Georgetown University
October 30, 2021

Introduction

This booklet has exercises which accompany the AllenNLP workshop given at Georgetown University on October 30, 2021. These materials assume familiarity with Python but no familiarity with PyTorch or AllenNLP. For the rest of the materials for this workshop including slides and code, please see `https://github.com/lgessler/allennlp-workshop`.

Exercises may be annotated with one [$\star$] two [$\star\star$] or three [$\star\star\star$] stars as an indicator of their difficulty. All starred exercises should be thought of as optional and will not impact your understanding if left unfinished.

Acknowledgements

Thanks to Georgetown University Computational Linguistics for funding snacks for the workshop, and to Georgetown University Department of Linguistics for providing space for the workshop. And thanks to the maintainers and creators of AllenNLP, of course!

Cover art: Modulation 62 by Julio le Parc, 1976.[1]

Back art: Stairway to the Studio by Charles Sheeler, 1924.[2]

[1]https://arthur.io/art/julio-le-parc/modulation-62

[2]https://arthur.io/art/charles-sheeler/stairway-to-the-studio

Contents

1 Introduction to PyTorch

Make sure you have cloned the workshop repository at `https://github.com/lgessler/allennlp-workshop` to your machine. Follow the instructions in the repository for setting up an Anaconda environment.

Using the Python environment you set up, run `jupyter lab` on your command line and open `/simple_classifier/1_pytorch.ipynb`. Note that each exercise corresponds to a single cell in the Jupyter notebook.

Exercises

1. What is the shape of the tensor `a`? Write code to print it.

2. The most common Python operators, such as the multiplication operator `*`, will work as you expect in an **element-wise** fashion. Print the product of `a` and `b`. Then, find a way to double the product of `a` and `b`, i.e. multiply each element of the product of `a` and `b` by 2. Your final answer should be `tensor([[8,12],[12,8]])`.

3. PyTorch tensors have many sophisticated tensor operations. A commonly needed one is matrix multiplication, which is provided as the tensor method `matmul`. Print the output of both `a * b` and `a.matmul(b)` to see how they are different.

4. Observe the code provided, which constructs a `torch.nn.Linear` module which can take 3-dimensional vectors in and yields 2-dimensional vectors out, and answer the following questions about the code:

   (a) What is the call to `torch.ones` doing?

   (b) What is `linear(output)` doing?

   (c) What is the shape of `output`?

   (d) [$\star$] In general, a Linear module computes $\mathbf{y} = \mathbf{x}\mathbf{W}^\top + \mathbf{b}$. In this exercise, how do the variable names `linear.bias`, `linear.weight`, `fours`, and `output` relate to the terms in this equation?

2 First Experiment

In this section, we will introduce you to a dataset of Yelp reviews.[3] Each item in the dataset is a representation of a Yelp review: `"text"` contains the review written by the Yelp user, and `"label"` is an integer value ranging from 0 to 4 that indicates whether the review awarded 1 star, 2 stars, etc.

The dataset is located at `/simple_classifier/data`. Training, validation (dev), and test splits are provided, both in full (600,000, 50,000, and 50,000 instances respectively) and in small subsets (5,000, 500, and 500). Data is provided in the JSON lines format, where each line in the file has a parseable JSON object.

You will use a very simple model to predict how many stars a reviewer awarded based on the text that they wrote.

Exercises

5. Use a text editor to open `/simple_classifier/data/train_5000.jsonl`, which contains 5,000 training instances. Look over the data and get a feel for it, e.g. the distribution of labels, the average length of a review, etc.

6. Use a text editor to open `/simple_classifier/configs/base.jsonnet`. Take a very brief look, noting that the paths to the training and validation data are given at `"train_data_path"` and `"validation_data_path"`.

7. Open `/simple_classifier/simple_classifier/model.py` and take a brief look. Note that the model expects an embedder and an encoder as input, and reports accuracy as its evaluation metric.

8. On your command line, navigate to the `/simple_classifier` directory of the repository. (**Note**: it is **very** important for your working directory to be exactly this.) We will now train and evaluate our model.

   Note that in between runs of your model, you will need to delete your serialization directory (`model`).

   (a) Begin training by invoking the train command.
   For MacOS/Linux:
   ```
   allennlp train configs/base.jsonnet -s model
   ```
   For Windows:
   ```
   allennlp train configs\base.json -s model
   ```
   Observe the output as your model runs.

   (b) When training is complete, note the accuracy your model achieved on the development set.

[3] `https://huggingface.co/datasets/yelp_review_full`

(c) Evaluate your model on the held-out test set using the evaluate command:

For MacOS/Linux:

```
allennlp evaluate model/model.tar.gz data/test_500.jsonl
```

For Windows:

```
allennlp evaluate model\model.tar.gz data\test_500.jsonl
```

(d) How different are the accuracies you achieved on the validation set and the test set?

9. When your model was trained, information about the accuracy and loss metrics was recorded. On your command line, start TensorBoard by executing `tensorboard --logdir model` and open it in your browser (usually, `http://localhost:6006`). Inspect the graphs: the y axis is either loss or accuracy, and the x axis is either batch number or epoch number.

10. [$\star$] Modify the `YelpReviewJsonLinesDatasetReader` so that labels can only be `"good"` or `"bad"`, where a review is `"good"` if the star rating is 4 or 5 (i.e., integer values 3 or 4 in the original JSON format) and `"bad"` otherwise. Re-run your experiments. Did you need to change any code outside of your `DatasetReader`? Has performance changed much?

    (**Note**: be sure to comment out your new code after you've completed this exercise.)

3 The Full Experimental Cycle

In this exercise section, we will take a look at the most important modules that are used during a training loop. Make sure your Jupyter lab process is running. Open `/simple_classifier/3_experimental_cycle.ipynb`.

Exercises

11. Step through the notebook's code, taking time to read code and output at each step.

12. Under "Vocabulary Setup", explain the output under "Label mapping:". What is the significance of this dictionary? (Recall that in the original dataset scores are given as integers between 0 and 4, and that the data reader we implemented transforms these into strings `"1"` to `"5"`.)

13. [⋆] The `print_probs` method takes an instance and shows the model's predicted probabilities for each class. (Note that we're using the `Model`'s `forward_on_instance` method, which is capable of batching and tensorifying on the fly.) Modify it so that it also prints the class which had the highest probability.

14. [⋆⋆] `Vocabulary` namespaces can be padded or unpadded. If a namespace is padded, two special tokens `@@UNKNOWN@@` and `@@PAD@@` will always be present in the namespace. `@@UNKNOWN@@` is used when a never-before-seen value is seen after training, and `@@PAD@@` is used when a sequence inside a batch is shorter than the maximum length of all sequences in the batch. By default, all namespaces except those which end in `tags` or `labels` are padded.

    (a) Use the method `get_token_index(token, namespace)` on `vocab` to find the index of a word that surely isn't in the dataset, like `"qweQWEqwe"`, for the `"tokens"` namespace. What is the return value? What is the significance of the return value?

    (b) Do the same for the token `"unseen_label"` in the `"labels"` namespace. Is the behavior the same as before or different? If it is different, why so?

    (c) Why not just pad all namespaces? (*Hint:* some modules inside a neural network will have their dimensions set to the size of a vocabulary, e.g. a classification layer whose number of output units will be equal to the number of classes.)

15. [⋆⋆] Use your favorite Python IDE with `debug.py` to debug a training process. Step through the code being executed and try to identify at a high level what's happening at every step in the code.

4 Fields

Open the next notebook at `/simple_classifier/4_fields.ipynb`.

Exercises

16. Observe the output of the different tokenizers. What is the difference in behavior between the whitespace tokenizer, the letters numbers tokenizer, and the SpaCy tokenizer?

17. Indexers use a vocabulary by looking up tokens within a given namespace in that vocabulary and returning the associated number. Consider the code given.

    (a) Does the character indexer look like it's working properly? What's wrong with it, and why is this happening?

    (**Hint**: use `vocab.get_token_from_index(index, namespace)` if it's helpful.)

    (b) Modify the code that initializes `reader` so that the problem will be fixed.

    (**Hint**: look at the classes that were imported at the beginning of this cell.)

18. [⋆] The `Embedding` class (a subclass of `TokenEmbedder`) has an optional parameter `pretrained_file` which accepts a path to pretrained word embeddings. Set this parameter to `https://allennlp.s3.amazonaws.com/datasets/glove/glove.6B.50d.txt.gz` to use pretrained GloVe embeddings and compare the results to those you get when not using pretrained embeddings.

19. [⋆⋆] Add a new field, `"text_length"`, to the dataset reader and model. The field should have an integer representation of the number of tokens that are in `"text"`, and it should be combined with the output of the encoder before `self.classifier` is called using `torch.vstack`. Refer to the `Field` documentation in order to choose an appropriate subclass: `https://docs.allennlp.org/main/api/data/fields/field/`.

    (**Hint**: you will need to add a new argument to the model's `forward()` method, and you will need to make a change to how `self.classifier` is initialized in the model's constructor. Remember that modules need to know the size of their inputs and outputs ahead of time.)

20. [⋆⋆⋆] Implement and tinker with stacked character embeddings.

    (a) Add character embeddings to the model, using the config shown in the slides for this section as a guide.

(b) Force characters to use the same `Vocabulary` namespace as regular tokens by setting `"vocab_namespace": "tokens"` in both the `TokenIndexer` and the `TokenEmbedder`'s `embedding` argument. Try running your model and comparing its performance.

(c) What is the bad thing that's happening here? Does it seem to be causing much of an issue for this situation? Could you imagine a situation where it would be a big issue?

5 Module Abstractions

Exercises

21. Swap the encoder we're currently using in our model for another one.

    (a) Consult the documentation for the `Seq2VecEncoder` abstraction: `https://docs.allennlp.org/main/api/modules/seq2vec_encoders/seq2vec_encoder/`

    (b) On the left-hand side, notice that listed under `seq2vec_encoders` there are many implementations of `Seq2VecEncoder` listed.

    (c) Click on `pytorch_seq2vec_wrapper` and then on the right-hand side, notice that there are six models to choose from.

    (d) Click on one of these models and note its arguments. (The `GruSeq2VecEncoder` is recommended, but any of these will work.)

    (e) **Copy** your `base.jsonnet` (or `base.json`) file into a neighbor file, `alternate_encoder.jsonnet` or `alternate_encoder.json`.

    (f) Modify the `"encoder"` key of your model so that it will use the different `Seq2VecEncoder` implementation you identified in step (d).

    (g) Train your model again from scratch to ensure that your model is working. How does the performance compare?

22. Implement your own tokenizer.

    (a) Create a new file named `my_tokenizer.py` in the `simple_classifier` package.

    (b) Create an implementation of `Tokenizer` using whatever method you'd like, using `WhitespaceTokenizer` as a guide for how to implement `Tokenizer`: `https://github.com/allenai/allennlp/blob/main/allennlp/data/tokenizers/whitespace_tokenizer.py`

    (c) Open `/allennlp-workshop/simple_classifier/__init__.py` and add the line `from simple_classifier.my_tokenizer import *` to ensure your tokenizer will be properly registered.

    (d) Modify your config so that your tokenizer will be used, and try running your model or opening a Jupyter notebook to play with it.

23. [⋆] Do anything you'd like to try to increase the performance of your classifier.

(a) Try using pretrained contextualized word embedding models (like `distilbert-base-cased`), following the guide here:
`https://guide.allennlp.org/next-steps#1`
Be warned that this will run very slowly without a GPU.