Deep Learning for NLP

Assignment 4: CNN Sentence Representations from Natural Language Inference Data

Sharid Loáiciga; edited by Philipp Sadler; code by Patrick Kahardipraja

1 Introduction

In this assignment, you will be working with a system for Natural Language Inference (NLI) using sentence representations based on a CNN architecture. The starter code is inspired by the work presented in the paper *Supervised Learning of Universal Sentence Representations from Natural Language Inference Data* by Conneau et al. (2017).

2 Natural Language Inference – NLI

Identifying entailment and contradiction are essential parts of semantics or meaning, and as such they are crucial for common sense reasoning. In this assignment, we use the Stanford Natural Language Inference Corpus (SNLI) described in Bowman et al. (2015). The complete corpus consists of $570,152$ sentence pairs, but we'll be working with a subset of $100,000$ pairs.[1] Each example pair consists of a premise and a hypothesis, which are connected by one of three inference relationships: *contradiction, entailment* or *neutral*. Table 1 presents some examples of the data.

Note that each pair has five annotator labels and a consensus label. Please use the consensus label, i.e., the `gold_label`.

The task in this assignment is to create a sentence encoder to process both the premise and the hypothesis in order to use them to predict the inference relationship connecting them (Figure 1).

Premise	**Inference**	**Hypothesis**
A man inspects the uniform of a figure in some East Asian country.	*contradiction*	The man is sleeping.
An older and younger man smiling.	*neutral*	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	*contradiction*	A man is driving down a lonely road.
A soccer game with multiple males playing.	*entailment*	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	*neutral*	A happy woman in a fairy costume holds an umbrella.

Table 1: Examples reported in Bowman et al. (2015).

[1] The SNLI corpus is available at `https://nlp.stanford.edu/projects/snli/`

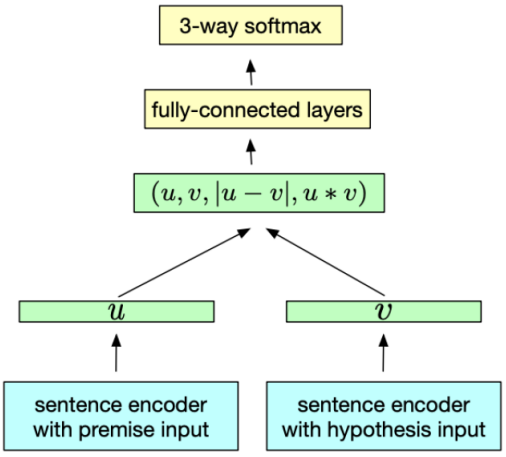

Figure 1: General NLI training scheme as presented by Conneau et al. (2017).

2.1 Sentence representation

In the last assignment, you were working with different types of RNN as sequence encoders. The sentence encoder that we use here is a Siamese network, i.e., the *same* weights are used to encode both the premise and the hypothesis.

The sentence representation will be achieved by using 2 layers of 1D CNN, to extract representations of the sentences at different level of abstractions. At each layer, the representation u_i is computed by a max-pooling operation over the feature maps. The final sentence representation u is a concatenation of the representations at different levels of the input sentence. The original implementation uses a kernel with size of 3, stride of 1 (you are free to experiment with this hyperparameters). Both sides are zero-padded.

Important Hints:

- The authors use "maxpooling over time" which is achieved with the `max` operation. In contrast to that the PyTorch `MaxPool1d` is a "maxpooling over channels". What would happen, when you use this instead? Your implementation should use `max`.
- In the paper, the authors use convolutions with "same" padding. You can check if your implementation is correct by ensuring that the input and output dimension of the embeddings is the same after each layer. What would happen, when you do *not* apply "same" padding?

2.2 NLI Network

As can be seen from Figure 1, there are three methods applied to extract relations between the premise u and the hypothesis v:

(i) concatenation of the two representation (u, v);

(ii) absolute element-wise difference $|u - v|$; and

(iii) element-wise product $u * v$.

The vectors are then concatenated together before being passed through a simple FFN with 1 hidden layer followed by a softmax layer. Use dropout with $p = 0.1$ for regularization.

3 Getting started

First download and unpack the archive `assignment4.zip`. This results in a directory `assignment4` with the following structure:

```
- snli_1.0/
- config.yml
- model.py
- README.md
- requirements.txt
- run.py
- utils.py
```

You should start by installing the requirements using `pip3 install -r requirements.txt`. We use GloVe embeddings (Pennington, Socher, and Manning 2014), installed by following the command in the `README.md` file. The directory `snli_1.0` contains the SNLI training, development and test sets in JSON format. You can use the development sets to find the optimal hyperparameters. Note that you should expect > 70% accuracy on the test set.

4 Starter Code

Let's look at the main program in `run.py`. It ...

- parses the input arguments to train or evaluate the model. This is started by setting the `-RUNMODE` flag to `'train'`, `'val'` or `'test'`.
- reads the `config.yml` containing the hyperparameters.
- reads the SNLI dataset using a PyTorch `Dataset` class implemented in `utils.py`.

The `Dataset` can be accessed through its index, similar to a list, to get the premise, hypothesis and their relation. For example:

```
PREMISE: A soccer game with multiple males playing.,
HYPOTHESIS: Some men are playing a sport.
GOLDLABEL:entailment
```

is represented as:

```
tensor([3, 12, 14, 54, 76, 89, 91, 0, 0, ...]), tensor([5, 7, 80, 91, 3, 101, 0, 0,
...]), tensor([0])
```

The premise and hypothesis tensors are padded, as we only load examples with maximum length of 50 and 1D CNN does not accept inputs with varying length.

5 Good practices

In this assignment, the code reads from a `.yml` file for the hyperparameters configuration. This may seem like an unnecessary extra step, but it forces you to be more systematic and organized, both attributes that come in handy when conducting large scale experiments. You can define the batch size for dataloader, the dimension of FFN hidden layer, the input and output channels of the convolutional layers, the number of training epochs, and the learning rate here.

5.1 Overfitting a batch

One debugging technique is to overfit a single batch to see if the model runs correctly. This can avoid wasting time before using a large dataset. Here's a recommended video explaining the advantages:

`https://www.youtube.com/watch?v=nAZdK4codMk`

You can set the `-DEBUG` flag when you are training the model to overfit a single batch.

6 Submission

This time we want you to zip your whole project with the changes made to `model.py` and `run.py` (and potentially including result files).

The ZIP file name should be `assignment4_LASTNAME.zip` and not larger than 20MB. For example, Sharid will name the file to `assignment4_LOAICIGA.zip`.

Upload your `assignment4_LASTNAME.zip` file on Moodle.

References

Bowman, Samuel R., Gabor Angeli, Christopher Potts, and Christopher D. Manning (Sept. 2015). "A large annotated corpus for learning natural language inference". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 632–642. DOI: 10.18653/v1/D15-1075. URL: https://aclanthology.org/D15-1075.

Conneau, Alexis, Douwe Kiela, Holger Schwenk, Loıc Barrault, and Antoine Bordes (2017). "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Association for Computational Linguistics, pp. 670–680. DOI: 10.18653/v1/d17-1070. URL: https://doi.org/10.18653/v1/d17-1070.

Pennington, Jeffrey, Richard Socher, and Christopher Manning (Oct. 2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.