Week 1: "Noisy" data

All the referenced files are available on LearnIt in the archive `assignment1.tar.gz`. The goal of this assignment is to:

- Get (re-)acquainted with classification problems in NLP
- Evaluate an analyze the result of an NLP experiment
- Implement a new solution to improve an existing classifier for social media data

1 Baseline

For this assignment we have provided two datasets; the Stanford Sentiment Treebank (SST) and data from SemEval-2013 Task 2: Sentiment Analysis in Twitter. Both of these datasets have been pre-processed to be in a similar format. The data from the SST dataset is relatively "clean" data, and is taken from review websites. The data from SemEval2013-2 is taken from Twitter, and can be expected to be more "noisy".

(a) Inspect the development datasets of both domains. Are there any systematic differences observable in the input texts?

(b) Inspect the provided code in `logres_clas.py`. What is used as input features for the classification?

(c) Run the baseline provided in `logres_clas.py`, train it on `sst.train` and evaluate it on `sst.dev` and `semeval2013.dev` using accuracy. How large is the performance difference between the in-domain review data and the Twitter data?

(d) It is well known that unknown words are problematic for NLP systems. One way to circumvent this, is to also incorporate character level information. Adapt the classifier so that it uses character 3-6 grams and word 1-2 grams. To do this you need to create two vectorizers and combine the results into a single feature space. You can use the SKLearn FeatureUnion (`https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html`) for this. Does the performance increase on the in-domain data? How about the out-of-domain data?

2 Approach 1: convert input data

The normalized version (from MoNoise) of the development data is included in `semeval2013.dev.normed`. Now evaluate the performance of the model that uses both words and characters as features from assignment 1 on this data.

(a) What is the performance on this dataset? Was normalization beneficial?

(b) Now find the cases where the model using normalization predicted correctly, and the model not using normalization predicted incorrectly. Inspect some of the differences in the input texts between the original and the normalized version. Are there any specific types of normalization that are prevalent?

3 Approach 2: convert training data

(a) Create a version of the training data where for each character one of 4 things can happen:

- It doesnt change: 91%
- A character is inserted after this character 3%
- The character is removed 3%

- The character is swapped with the previous character 3%

(b) Train on the new training data; and evaluate on the original (not normalized) dev set. How did it impact performance?

4 Bonus

(a) Automatically create alternations of the training data in `sst.train`.

- You can take inspiration from the Ùfal paper available at: `https://aclanthology.org/2021.wnut-1.54/`, or the GenERRate paper: `https://aclanthology.org/W09-2112.pdf`.
- You can create as many variations of the training instances as you find useful. You can also include the original training data.
- Note that the goal is **not** to improve the model hyperparameters or add annotated training data.

(b) Train the model on the new training data, you can use the original (not-normalized) dev set for development.

(c) Run the model based on your best training data on the test data (`semeval2013.test_masked`) and attach your best prediction to your uploaded solution on LearnIt.

Week 2: RNN and Multi-task Learning

The goal of this assignment is to:

- Get (re-)acquainted with BiLSTMs and PyTorch.
- Learn to adapt an NLP model to a new domain through language modeling.
- Implement a simple method for neural multi-task learning: adding a decoder head for each task.
- Compare performances of different settings, and think about how to interpret these results.

5 Baseline

The datasets are the same as for the week 1 part. The baseline model is an LSTM, familiarize yourself with the code in `bilstm.py`

(a) What does the input data look like after it has been returned from the `readFile` function? (i.e. what is the shape of `train_features`?, what do the values represent?)

(b) What is the shape of the `log_probs` as returned by the forward function of the model? (not just the numbers, what do they represent?)

(c) In the final feedforward layer, the input dimension is set to `lstm_dim*2`, why is the multiplication by 2 necessary?

(d) Evaluate a model trained on SST on both the in-domain data and the SemEval2013 dev data.

(e) How many unknown words are there in both dev datasets?

6 Approach 1: using word embeddings for adaptation

Initialization with pre-trained embeddings can be used to inject "knowledge" about a "language" into a model. In this assignment, we are going to use word embeddings trained with word2vec on Twitter data (`http://www.itu.dk/~robv/data/embeds/w2v-twitter.txt.gz`). More information about how these embeddings are trained can be found on: `https://robvanderg.github.io/modeling/twit-embeds/`

(a) Load the embeddings, and use them to initalize the `self.word_embeddings` in the `Classificator model`.

- You can use the `nn.Embedding.from_pretrained` function (`https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html#torch.nn.Embedding.from_pretrained`)
- Note that this expects a 2-dimensional torch tensor, the first dimension matches the number of words (4,428,756), and the second dimension the size of the embeddings (100).
- The indices in the original matrix should match the word indices you use as input, so the `Vocabulary` class needs to be adapted. The easiest way to do this is probably to just create the word2idx and idx2word based on the embeddings file (note that the Unknown token is [UNK], and is at position 0 already)
- The embeddings file is in text format, the first line contains the dimensions, followed by a word with its representation per line:

```
4428756 100
[UNK] 0.004003 0.004419 -0.003830 -0.003278 0.001367 0.003021 0.000941 0.000211 -0.003604 0.00
```

(b) Is the number of unknown words reduced compared to the baseline (without pre-trained embeddings)?, What are the remaining unknown words?

(c) Evaluate the performance when training on SST and evaluating on SemEval2013; are the embeddings beneficial?

7 Approach 2: Multi-task Learning on Auxiliary Tasks

Sarcasm detection is relevant for sentiment analysis, as it can invert the sentiment label. Hence, we will test the hypothesis whether we can use sarcasm detection as auxiliary task to support our sentiment analysis system.

We will use data from the 2020 Sarcasm Detection Shared Task (`https://github.com/EducationalTestingService/sarcasm/`), we converted it to the same format as the sentiment data in `figlang.train` and `figlang.test`.

(a) Add a feedforward classification layer to the existing model that can classify sarcasm. This is a simple method to implement multi-task learning, with the maximum amount of sharing, you can see a diagram of the resulting model in Figure 1.

(b) Implement data reading for the sarcasm task; make sure the word embeddings (and word indices) are shared across tasks, but the label indices should not be shared!

(c) Adapt the training procedure. You can do this by adding another for loop within each epoch in which you traverse through the batches of the second dataset. You can reuse the optimizer and the loss from the first task. Make sure you use the right predictions and compare to the matching gold labels for each task.

(d) Evaluate performance; is the sarcasm task beneficial?, why would this be the case?

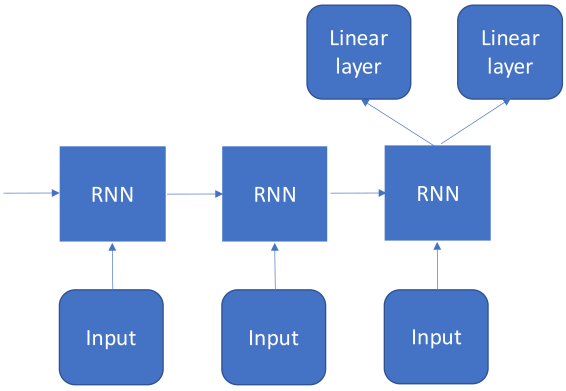

Figure 1: Simple multi-task model that performs two tasks simultaneously.