LTAT.01.001 Homework 4

Sequence Tagger in AllenNLP

Elizaveta Korotkova

Character-level RNN

The model is used to classify last names, determining to which language they belong. In the training data we have 18 files, one per language, each containing a list of last names of that language.

Our model is a character-level recurrent neural network: we will pass names through an RNN, treating them as sequences of characters, each character represented by its low-dimensional embedding. The final output of the RNN is then transformed using a linear layer and a softmax layer to obtain predicted class scores.

Code

In the Github repository (github.com/lisskor/allennlp_homework4) there are three modules: `reader.py`, `model.py`, and `predictor.py`.

Reader

`NamesDatasetReader` is needed to read in data from files and transform into a format AllenNLP can work with. The `read` method accepts a path to a directory containing all the necessary files. It gets the class names from filenames in the directory, iterates over all files in the given location, turns each name into plain ASCII (following the PyTorch tutorial), and yields names one by one after transforming them into `Instance`s. The transformation is done by the `text_to_instance` method: it accepts a token and optionally a class label, and returns an `Instance` with fields `name` (a `TextField` containing the name), `name_characters` (a `TextField` containing the name indexed by characters), and possibly `label` (a `LabelField` storing the class label).

Model

`NamesClassifier` is the model itself. It accepts instances produced by `NamesDatasetReader`. It applies zero-padding to the instances, so that all instances of a batch will be represented by vectors of the same size. It then turns character indices into character embeddings, and uses an RNN unit as a `Seq2VecEncoder` (since we only need one output per sequence, and not one output per character) on the obtained sequence of embeddings. The output of the RNN is passed through a linear layer to obtain class scores, and through softmax to obtain class probabilities and cross-entropy loss. Hidden state is fed to the RNN unit at the next step.

Predictor

`NamesPredictor`'s method `predict_json` will take a dictionary (e.g. `{'name': 'Belucci'}`), turn it into an `Instance`, and predict a class for it (e.g. 'Italian').

Results

I trained a model using an LSTM with hidden size 6, character embeddings of dimension 6, bucket iterator, batch size 32, SGD optimizer with learning rate 0.1 for 50 epochs (see `config.jsonnet`). The model was both trained and validated on the whole dataset available (which is obviously suboptimal, but I left it that way for simplicity). I was able to achieve validation loss of 0.869 and validation accuracy of 0.755. The trained model is in `model.tar.gz`.