Anastasiia Kysliak

Report for the practical assignment

Task 1

A model with stacked bidirectional LSTMs followed by softmax functions for multi-task learning was suggested and implemented (with 3 biLSTMs layers, dropout=0.2). Its Input represents one complete sentence, with each token embedded by pre-trained word embeddings (WE). For slot predictions, outputs of the last biLSTM layer at each time step are linearly transformed and passed through a softmax for label classification. For intents, hidden states of the biLSTM in the last layer after processing the whole sequence in both directions get concatenated, linearly transformed, and passed through another softmax. Cross-entropy losses computed in both cases get summed in a joint loss for backpropagation. The learning was performed on-line, with SGD optimizer. GloVe (6B) and FastText WE from torchnlp.word_to_vector package were used (both with dimensions 300), without fine-tuning. Advantages of this choice are reasonable memory size needed and computational speed, plus useful functionality like initialising embeddings for out-of-vocabulary words with zeros. However, these WE are not uncased, therefore, many words were considered unknown. Nevertheless, models with both WE could successfully learn, see Figures 1 and 2. After 10 training epochs, the results on the test set were: **0.938** for slots, **0.981** for intents for GloVe, **0.929** for slots, **0.98** for intents for FastText. As evident from Figures 1 and 2, GloVe constantly outperformed FastText during training, which also correlates with its better results on the test set. **Improvement suggestions**: use uncased WE; allow fine-tuning of WE before and/or during training; tune hyperparameters (weights for summed losses, number of biLSTM layers, dropout, etc.); incorporate early stopping criteria for training: when no improvements observed on the development set during N evaluations; try other optimisers, process data in batches.

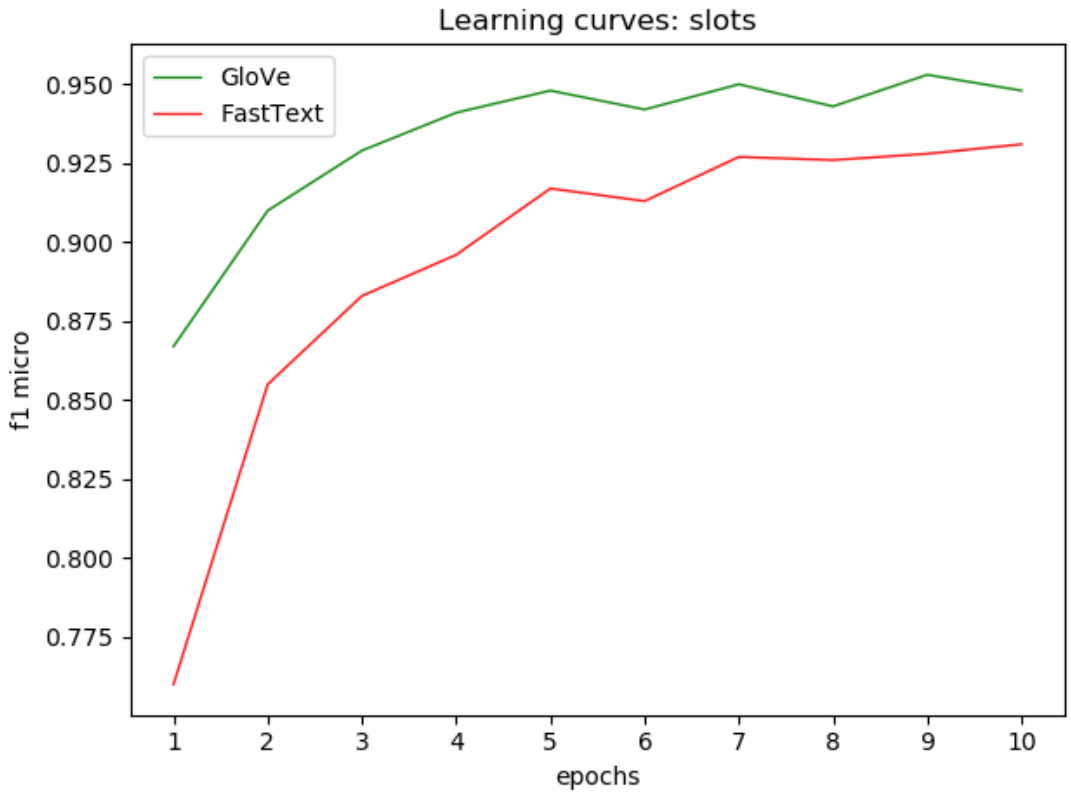

Figure 1 F1-score for slots on the development set after training epochs

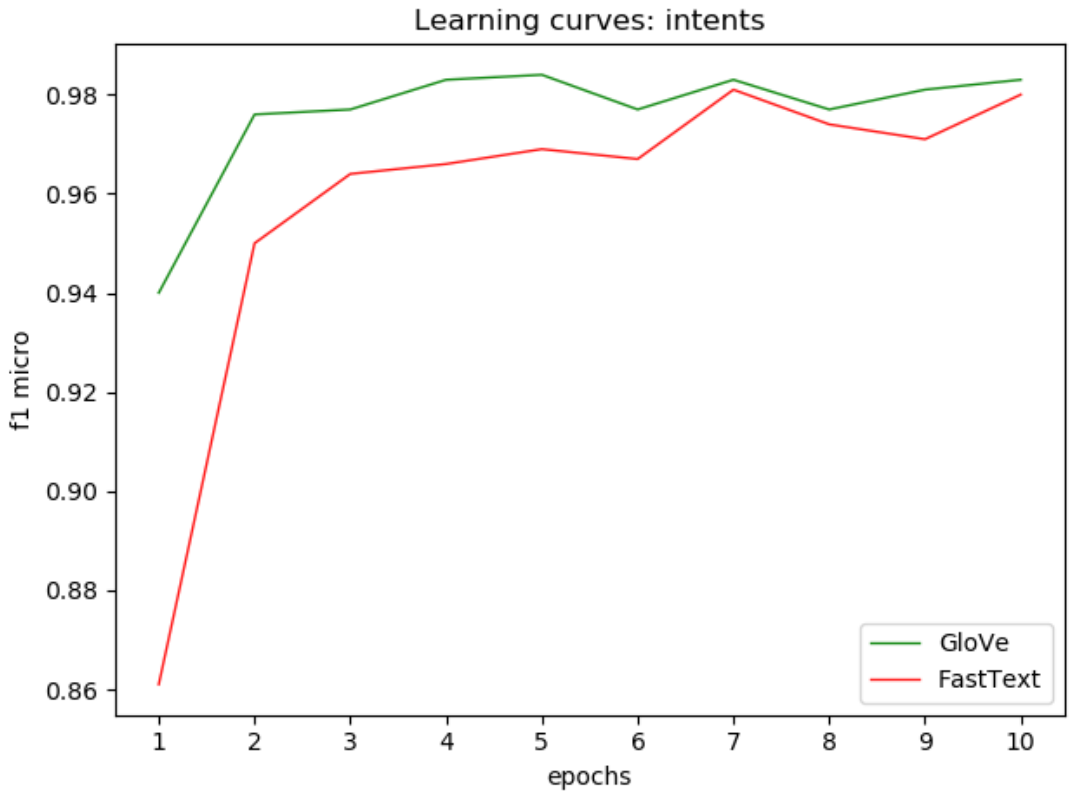

Figure 2 F1-score for intents on the development set after training epochs

Task 2

Classification was performed with kNN clustering for the PlayMusic domain, with same GloVe and FastText WE as in Task 1. The following input representations were considered: unigrams (only the embedded target word), bigrams with left context, bigrams with right context, trigrams with context words around the target. Embeddings of words within higher-order n-grams were concatenated, "#######" represented the beginning and end of sentences (absent in training data and present in WE, was expected to have distinctive embeddings). Euclidean distance was calculated between datapoints. Settings with 1, 5, 10, 20 nearest neighbours were considered, with and without data augmentation. Without augmentation, only twenty_proto.txt or fifteen_proto.txt for PlayMusic were used. With augmentation, examples from corresponding proto-files for AddToPlaylist were added: only those with target slots shared between both domains ("O", "playlist", "artist", "music_item"), to avoid extra noise. All parameter settings were evaluated with 10-fold cross-validation on the training data. Models based on n-gram counts (n=1…3) were used as baselines. Unigram baseline and another one with back-off strategy performed best (due to data sparsity). Micro-averaged f1-score was used for evaluation. Best results for models with WE were obtained with trigrams for K=1 in kNN. Figures 3 and 4 show that models performance degraded with higher K, apparently, due to data sparsity: distant neighbours within K nearest ones introduced noise. In cross-validations and on the test set, FastText performed generally worse than GloVe in identical settings, even worse than a back-off baseline for K=20. Data augmentation was often beneficial, especially for twenty_proto setting (with more training examples). However, it was less advantageous for fifteen_proto. For each WE, results for twenty_proto and fifteen_proto settings were competitive. **Improvement suggestions**: fine-tune WE on the training corpora for the target domain, including train.txt; use uncased WE; reduce vector dimensionality, e.g with PCA, to increase datapoints density.

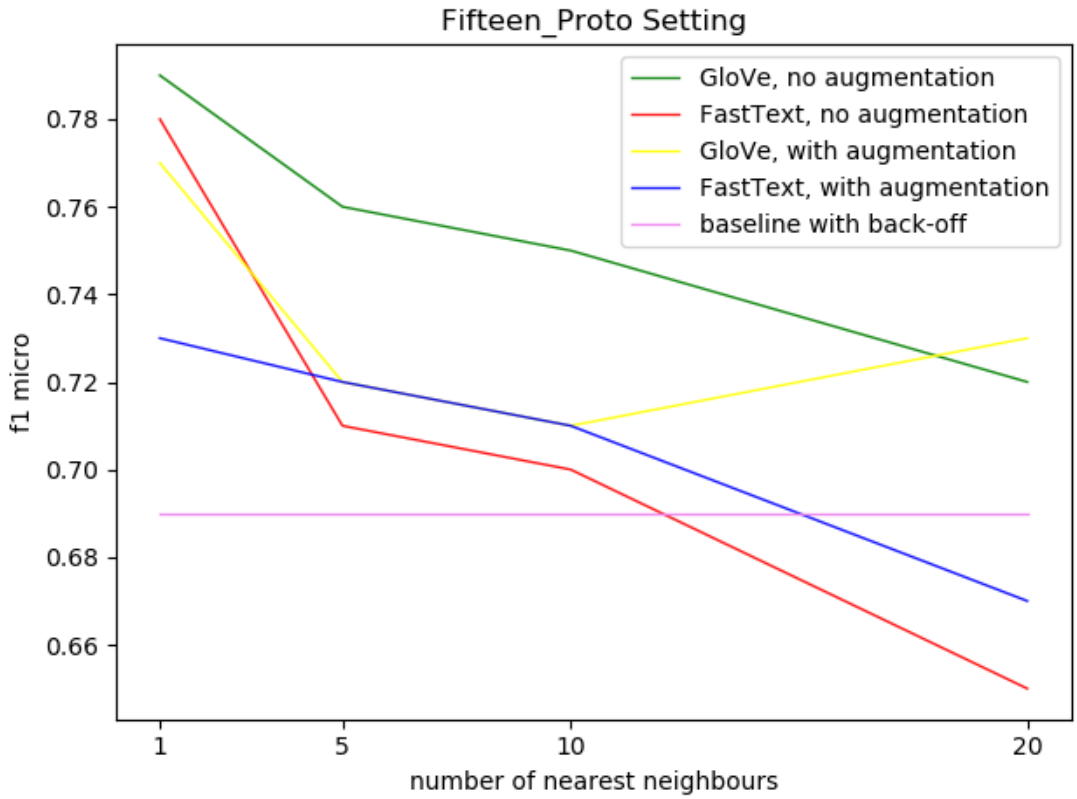

Figure 3 Results for different K in kNN

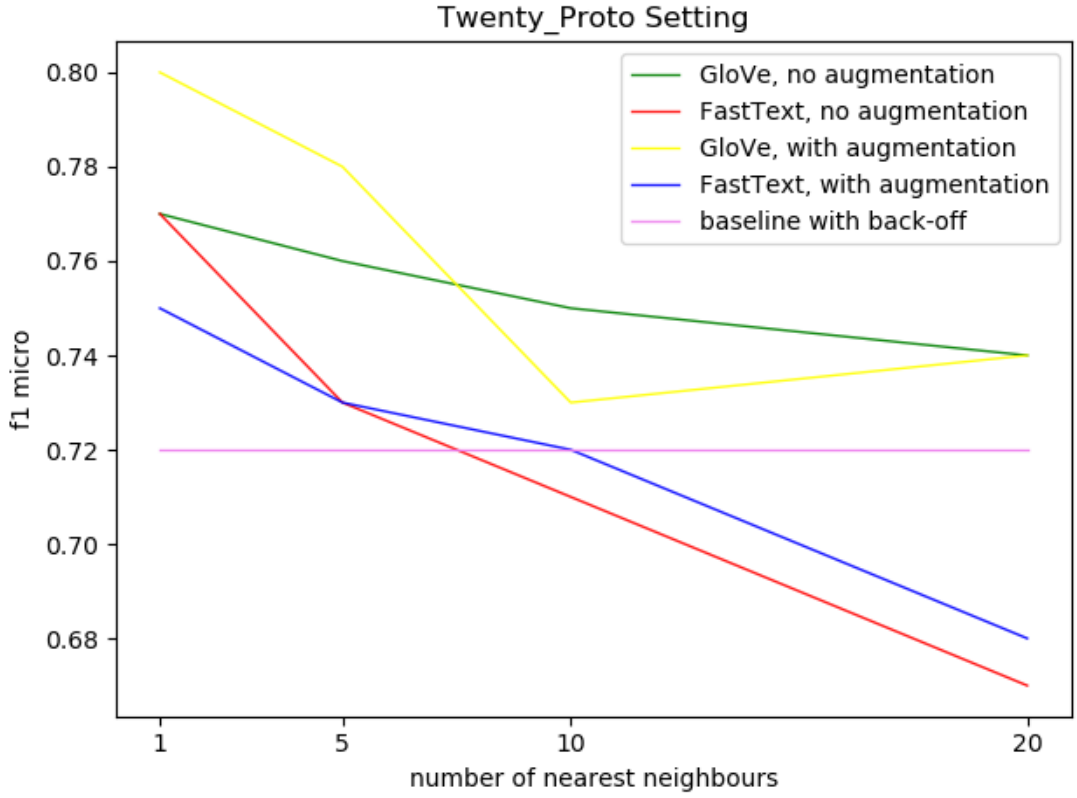

Figure 4 Results for different K in kNN