Data

My LSTM model has been trained on 16k POS tagged sentences on the ud-english-treebank, UD_English_EWT

Data has been preprocessed by converting to lowercase, and making all sentences a fixed length of 20, either by post-padding or truncation. The mean length of sentences in the training set is about 12-15, so enforcing a length of 20 covers a large portion of the dataset whilst avoiding the model overfitting on always detecting padding. Without this, the model learns that it recieves a high score randomly and frequently inserting padding tags due to the massive number of padding tags in each sentence on average.

Architecture

1. Embedding layer converting vocabulary indices into e-dimensional embeddings
2. LSTM model with an h-dimensional hidden state, h-dimensional cell state
3. 2 fully connected layers to classify the LSTM output into tags (hxh, hx(num of tags))
4. Using an SGD optimiser and cross entropy loss. Reported losses are the average cross entropy loss over all sentences

After tuning parameters, the best results are:

- Params
    - Embedding size (e) = 100
    - Hidden state size (h) = 150
    - Learning rate = 0.01
- Errors
    - Train error = 0.4523
    - Dev error = 0.0331
    - Test error = 0.0322

This set of parameters obtained an average accuracy of 81% on test data. The full sklearn report is reported below.

Other tried sets of params

- Single fully connected layer
    - Params
        - Embedding size (e) = 50
        - Hidden state size (h) = 50
        - Learning rate = 0.01
    - Errors
        - Train error = 0.576
        - Dev error = 0.037
        - Test error = 0.037

    - Params
        - Embedding size (e) = 100
        - Hidden state size (h) = 100
        - Learning rate = 0.01
    - Errors
        - Train error = 0.4523
        - Dev error = 0.0331
        - Test error = 0.0322

    - Params
        - Embedding size (e) = 50
        - Hidden state size (h) = 100
        - Learning rate = 0.01
    - Errors
        - Train error = 0.518
        - Dev error = 0.035
        - Test error = 0.0338

- 2 FCS

    - Params
        - Embedding size (e) = 50
        - Hidden state size (h) = 50
        - Learning rate = 0.01
    - Errors
        - Train error = 0.576
        - Dev error = 0.037
        - Test error = 0.037

    - Params
        - Embedding size (e) = 100
        - Hidden state size (h) = 100
        - Learning rate = 0.005
    - Errors
        - Train error = 0.469
        - Dev error = 0.0344
        - Test error = 0.0334

    - Params
        - Embedding size (e) = 100

        - Hidden state size (h) = 100
        - Learning rate = 0.01
    - Errors
        - Train error = 0.45
        - Dev error = 0.033
        - Test error = 0.0322

sklearn.metrics.classification results on the chosen params

	precision	**recall**	**f1-score**	**support**
_unk	0.00	0.00	0.00	0
_pad	0.99	1.00	1.00	12902
INTJ	0.00	0.00	0.00	96
PRON	0.96	0.91	0.94	1928
VERB	0.88	0.55	0.67	2297
DET	0.91	0.97	0.94	1592
NOUN	0.56	0.89	0.69	3309
ADV	0.95	0.46	0.62	979
PUNCT	0.95	0.99	0.97	2169
AUX	0.78	0.98	0.87	1323
PART	0.55	0.40	0.46	564
ADP	0.84	0.84	0.84	1632
PROPN	0.82	0.02	0.03	1292
NUM	0.50	0.00	0.00	403
CCONJ	0.81	0.98	0.89	619
SCONJ	0.98	0.26	0.41	401
ADJ	0.88	0.35	0.50	1306
SYM	0.00	0.00	0.00	61
_	1.00	0.01	0.03	275
X	0.00	0.00	0.00	112
accuracy			0.81	33260
macro avg	0.67	0.48	0.49	33260
weighted avg	0.88	0.81	0.81	33260

Analysis

Increasing the size of the hidden dimension seems to increase the accuracy of the model. This probably scales with the number of words in the training set sentences. Other optimisation methods like Adam may perform better. Adding more classification layers has a small increase in performance, perhaps deeper classification layers perform better. The embedding layer may not have enough input data to sufficiently form relations. Common bigrams of determiners like PROPN ADP PROPN are learnt easily, but the model does not predict nouns and verbs that well. This can be attributed to the embedding layer not being trained sufficiently.