Named Entity Recognition

With bilstm

$$\begin{pmatrix} \underline{\mathbf{i}} \\ \underline{\mathbf{f}} \\ \underline{\mathbf{o}} \\ \underline{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{sigm} \\ \tanh \end{pmatrix} \left(\begin{pmatrix} \mathbf{x}_t \circ \mathbf{z}_{\mathbf{x}}^t \\ \mathbf{h}_{t-1} \end{pmatrix} \cdot \mathbf{W} \right)$$

Carmel Eliav

NLP

Matches words, embedded with word2vec to news item tags - in sequence.

data

1. Tagged news text with 12101 lines (11100 words).
2. Word vecs with 300 features.

Models and hyperparameters

1. Bilstm with a single layer.
2. optimizer= ‘adam’
3. Loss= negative log likelihood
4. Learning rate= 0.001
5. Weight decay=1e-08

Performance

Epochs	Training accuracy	Test accuracy
10	100%	46%
25	100%	45%
50	100%	45%

RESULTS

Due to lack of data my model suffered from over fitting.even with a dropout of 0.5, By the 10’s epoch the training accuracy was nearly 100% and the test accuracy was only 30%.

By using regulazations methods (learning rate scheduler, weight decay) and shuffling the data before each epoch i managed to increase the test result up to 47%.

Methodology

Overview

Network: 150-unit BiLSTM with dropout -> Log_Softmax

Loss: negative log-likelihood

Optimizer: Adam

Preprocessing, Features, and Data Format

1. Encode the words (X) a. Assign each word a 300-length word2vec representation, from the provided file. b. If no vector exists for a word, sample a 300-length vector from a multivariate normal distribution(for embeddings) .
2. Encode the tags (y) a. Read through the tagged news data to get the number of tags (classes). b. Add 'nil' class for unknown data. c. Assign each class an id. d. Assign each class a 1-hot vector and id, length= # classes, where the tag id is the index of the 1.
3. Assemble the data a. Read the tagged data from the provided file. b. For each (sentence, tags) example, i. Map each word in sentence to a vector from 1). ii. Map each tag to an id from 2).

- Features in a frame of X = 300
- Features in a frame of y = 10

- all_X_data.shape = [n_samples,sequence length,,300]
- all_Y_data.shape = [n_samples,sequence length, 10]

NOTE On Prediction

Unknown words encountered during prediction are assigned a unit, 300-length vector from a multivariate normal.

Model Details and Design Choices

- This problem is known as sequence labelling. Given a (variable) length sequence of X's, for each element X in the sequence, predict a discrete y that is associated with X.
- Missing Words Embedding:
    - A 3rd of all the words were missing vector representations - this is a significant amount of missing data. It is important to include this data in the model, and thus encode them.
    - Word2vec initializes each word vector embedding as a sample from the multivariate normal distribution.
- Sequence Length:
    - Each sequence has a different length and is single batch.
    - The data loader loads one sequence at a time.
    - The network learns to associate nil frames with nil classes
- Recurrent Unit Selection:
    - LSTMs and GRUs prevent vanishing and exploding gradients in RNNs by gating the activation signals.
    - LSTMs and GRUs also learn long term dependencies, which is helpful
- Bi-Directional:
    - The news tags depend on the words that came before and that come after, so lookahead ability is preferred.
    - BiDi-RNNS have 2 RNNS, one operating in each time direction and feeding into each out to facilitate this.
- Multi-class, Loss, and Softmax:
    - In this task I minimize the loss between the true and predicted label for a sequence. by using negative log likelihood loss function.
    - Multi-class classification in neural nets usually encodes the class as a one hot vector, then has an output unit for each possible class = each element of class vector. The class is then determined by which output unit / element of output vector has the highest strength.
    - I used Log_softmax on the output vector to induce a probability distribution.
    - The final recurrent layer of my model has 10 outputs (one for each class +

    nil) which is log_softmaxed. This layer is present at every time step (last layer in stack) to provide an estimate for every frame.

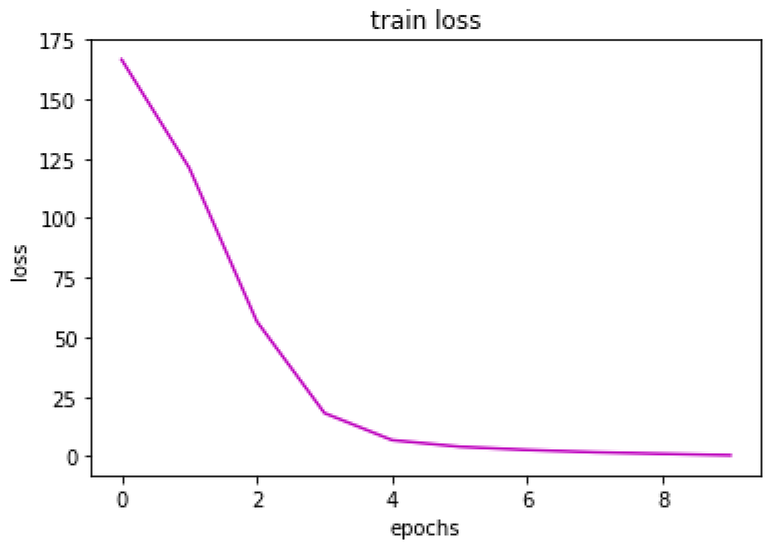
- Optimizer:
    - Adam accelerate convergence by looking at the history of the gradients (and other).
    - I used adam with weight decay of 1e-08 and learning rate of 0.001.
- Batch Size:
    - Each sequence is batch.
    - It means that after every iteration(sequence) back propagation is performed.
- Dropout:
    - My model suffered from over fitting so i used a dropout of 0.5.
    - Dropout of 0.2-0.4 seemed to benefit poor results.
- Regularization:
    - As mention the model suffered from over fitting so besides dropout i used weight decay of 1e-08, learning rate scheduler (ReduceLROnPlateau) with patience of 3 which sets the learning rate of each parameter group to the initial lr decayed by gamma every step_size
    - In another attempt to reduce the over fitting the data loader shuffles the data before each epoch.
- Train/Test split = 80/20

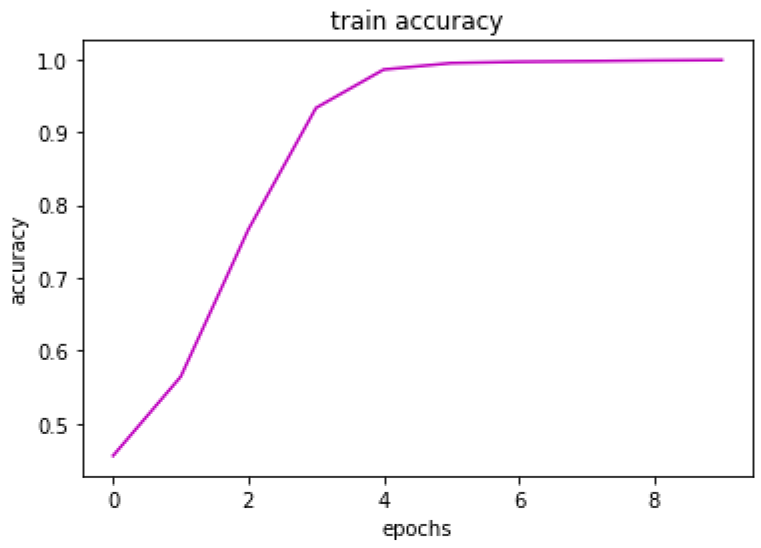
Notes

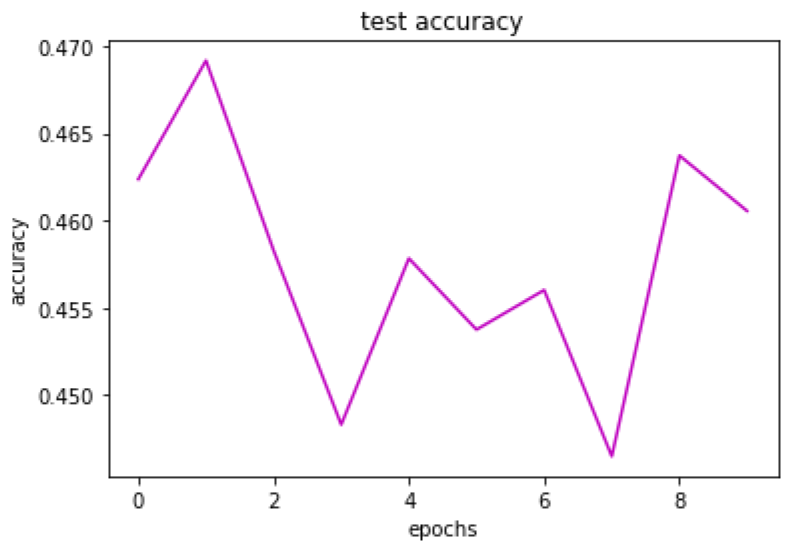
- This is my first time using pytorch.I learned pytorch to complete this challenge.
- This is my first time implementing recurrent neural networks. It was especially challenging since the problem involves variable length sequences, forward and backward dependencies. i used bidirectional LSTM layers to approach it.
- Eventually i couldn't resolve the overfitting problems even though i used varias regularization methods. The main cause was lack of data. (approximately 1000 sequences).

Experiments results:

Epochs: 10



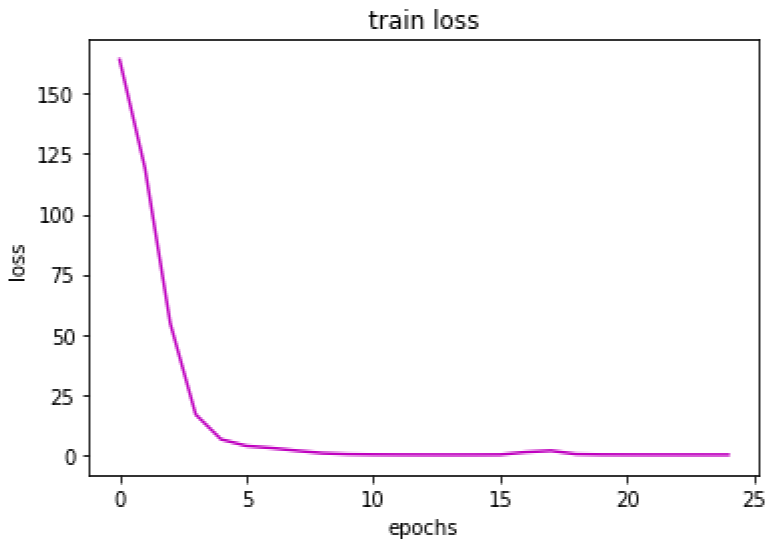


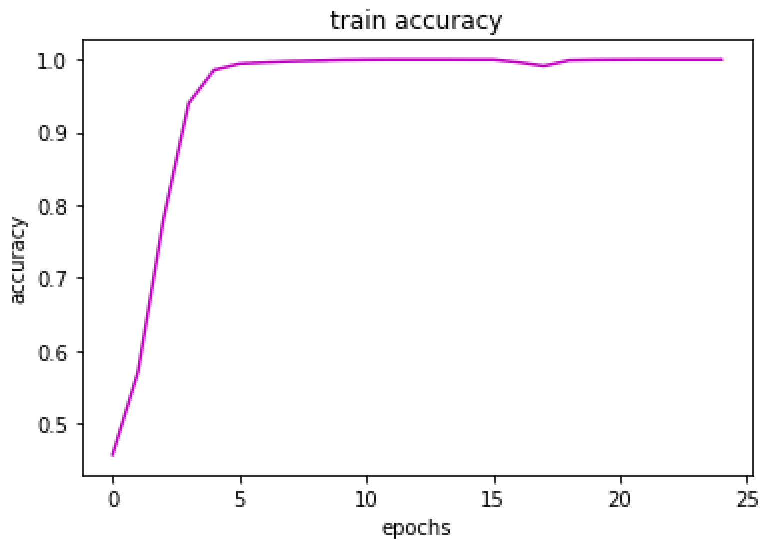


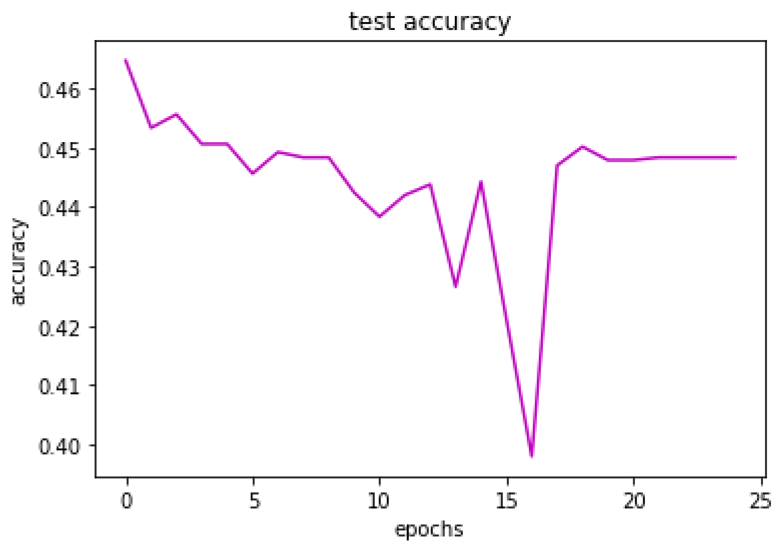
```
epoch 10 ouf of 10##

        Train batch[0/800] - loss: 0.000483  accuracy: 100.0000%(3/3)
        Train batch[100/800] - loss: 0.000073  accuracy: 100.0000%(14/14)
        Train batch[200/800] - loss: 0.012208  accuracy: 88.8889%(8/9)
        Train batch[300/800] - loss: 0.000097  accuracy: 100.0000%(11/11)
        Train batch[400/800] - loss: 0.000143  accuracy: 100.0000%(11/11)
        Train batch[500/800] - loss: 0.000223  accuracy: 100.0000%(9/9)
        Train batch[600/800] - loss: 0.000229  accuracy: 100.0000%(11/11)
        Train batch[700/800] - loss: 0.000059  accuracy: 100.0000%(14/14)
```

Train set: accuracy: 8873/8883 (100%)
Test set: Average loss: 5.5213, Accuracy: 1016/2206 (46%)

Epochs: 25


train loss
loss
epochs
0
25
50
75
100
125
150
0
5
10
15
20
25


train accuracy
accuracy
epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
5
10
15
20
25



```
epoch 25 out of 25##

          Train batch[0/800] - loss: 0.000041  accuracy: 100.0000%(16/16)
          Train batch[100/800] - loss: 0.000007  accuracy: 100.0000%(6/6)
          Train batch[200/800] - loss: 0.000048  accuracy: 100.0000%(14/14)
          Train batch[300/800] - loss: 0.000035  accuracy: 100.0000%(13/13)
          Train batch[400/800] - loss: 0.000011  accuracy: 100.0000%(6/6)
          Train batch[500/800] - loss: 0.000069  accuracy: 100.0000%(8/8)
          Train batch[600/800] - loss: 0.000006  accuracy: 100.0000%(8/8)
          Train batch[700/800] - loss: 0.000043  accuracy: 100.0000%(15/15)
```

Train set: accuracy: 8883/8883 (100%)
Test set: Average loss: 6.5480, Accuracy: 989/2206 (45%)