325069565 Idan Pogrebinsky 212778229 Ilai Avni

Pre-proccessing:

At first we break down the file into sentences, then filter the data, pass the words through the chosen word embedding, add padding and crop the sentences to be tensors of size 100x100 and save it to the disk. Additionally we batch the sentences with a custom function instead of using data loaders to avoid the overhead.

We keep a dictionary that's meant to translate POS to their corresponding index.

At training time the input to the model is a sentences tensor, of shape [batch_size, max_sentence_len, word_embedding_size], a POS indexes tensor, of shape [batch_size, max_sentence_len] and a sentences_len tensor, of shape [batch_size, max_sentence_len]

The forward

At first we pass the POS indexes thought the pos embedding layer, concat the embedded POS with the matching word embedding, and then pass the results to the Bi-LSTM.

Now we have a hidden state for each word, we generate all the possible pair combinations between the hidden states of the words, and pass all those pairs into a fully connected network that outputs an edge weight for each pair of words, with respect to the order of the pair.

Then all those weights are rearranged into a score matrix as the one that Chu-Liu-Edmonds function expects to get.

And then then we output those batched matrices, at size [batch_size, max_sentence_len, max_sentence_len]

Then we pass the model outputs through the given Chu-Liu-Edmonds function and return the decoded outputs.

Loss calculation:

We use per word cross entropy, and sum the losses over the sentences, perform the backward for each batch.

Training time:

the model takes about 35 seconds per epoch on our laptops, which is quick, and the results already start making sense after 2 epochs. Yet we give it X epochs to converge and reach its peak performance.

Detailed info about the model structure:

Word embedding: glove-wiki-gigaword-100

POS embedding: torch.nn.embedding layer with the embedding size of 50

LSTM:

We have a Bi-directional LSTM which consists of 4 layers, hidden size of 256, and an input size of 150 = 100(word-embedding-size) + 50(POS-embedding-size)

POSTagFC:

A network which consists of 3 fc layers:

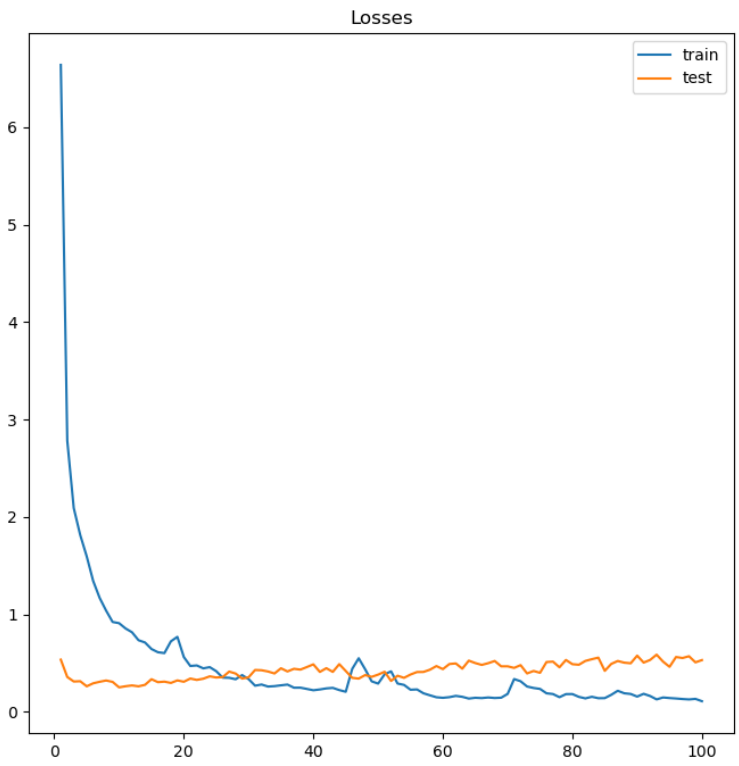
```
self.layer1 = nn.Linear(input_size, 2 * hidden_size)
self.layer2 = nn.Linear(2 * hidden_size, hidden_size // 2)
self.layer3 = nn.Linear(hidden_size // 2, output_size)
```

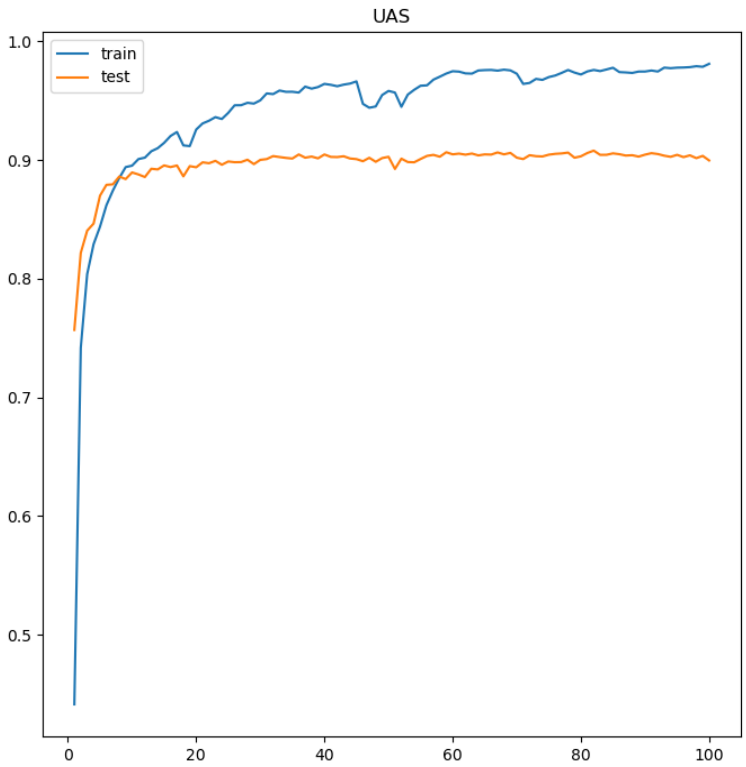
And uses a relu function between the layers and dropout during the training time.

The UAS score for the train set is: 96%

The UAS score for the test set is 91.1%

Loss and UAS Graphs





Changes for the competitive part:

We didn't make any changes for the competitive part.

But we did attempt to use 'optuna ' library, which is a library used for hyperparameter search, but after almost 10 hours of configuring it, we couldn't make it to find anything better than our initial model, so we gave up on that.

The work distribution:

We started together and programmed the core of the code together, and since the second day of work we split up and started taking turns on the code while frequently updating and consulting one another.

The dry questions were answered by Ilai and the report was written by Idan.