GitHub: https://github.com/nitaifingerhut/dependency-parser

1 Train

1.1 Batch implementation

PyTorch DataLoader does not support a batch of samples with different sizes, such as we have here (since each sample is a sentence, and sentences' lengths vary). Since batch optimization may improve training and convergence, we manually implemented it. We accumulated the gradients for each sentence and applied the back-propagation and optimization step only after viewing **batch_size** number of sentences.

1.2 Run time:

Basic model ∼ 1 hour for 100 epochs

Advanced model ∼ 1.5 hours for 100 epochs

1.3 Hyper-parameters

1.3.1 Basic model

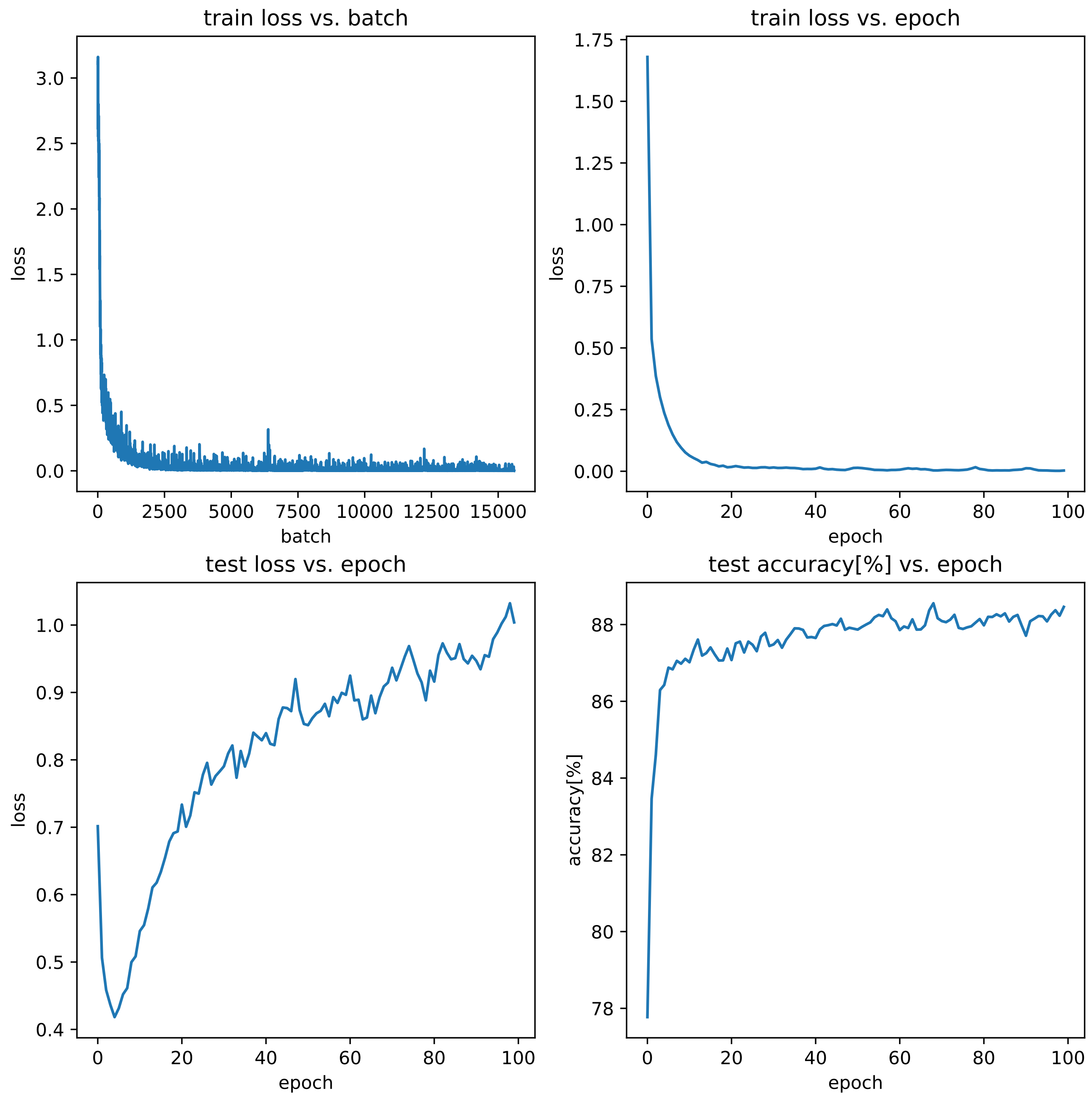
Parameter	Value
Larning rate	0.001
Bach size	32
Epochs	68
Dropout rate	0.1
Optimizer	Adam
Layers	2
Layers Dimension	128
Activation Function	tanh
Random word embedding dimension	200

1.3.2 Advanced model

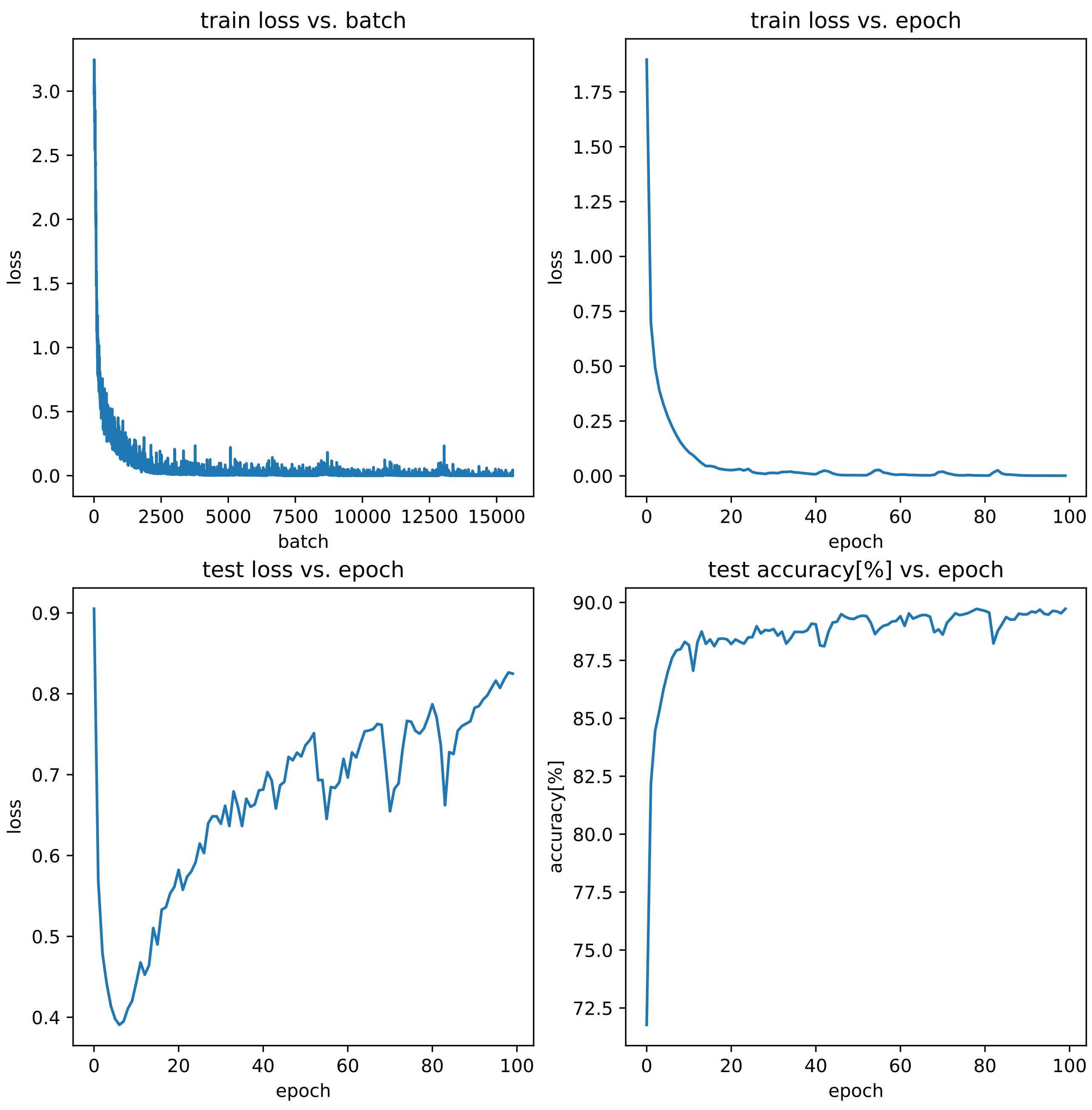
Parameter	Value
Learning rate	0.0003
Bach size	32
Epochs	99
Dropout rate	0.1
Optimizer	Adam
Layers	4
Layers Dimension	512
Activation Function	tanh
Pre-trained word embedding	glove.6B.300d

1.4 Graphs

Basic Model



Advanced Model



2 Test

2.1 Accuracy:

Basic model 88.55%

Advanced model 89.73%

2.2 Run time:

Basic model $<$ 1 minute

Advanced model $\sim$ 2 minutes

2.3 Accuracy gap

Most of the hyper-parameters are similar in both models, we found 2 ways to improve performance:

- Trained word embedding: when using a pre-trained word embedding similar/related words have similar word embedding compared to non-similar words. the main advantage of this setting is in the fact that similar words tend to appear in the same part of the sentence. if two words have the same role in the sentence the net needs to learn to allocate them to the same part of the sentence, it should be much easier if the embedding is fairly similar.
- Number of layers and layers dimension: one could look at the network structure as another Hyper-parameter and tune it to maximize performance. in the advanced model, we found 4 layers (of LSTM) with hidden states dimension of 512 works better than the structure proposed in the article of 2 layers and hidden states dimension of 128.

3 Competition

3.1 Main work and improvements

- Hyper-parameters tune: we conducted a big hyper-parameters search by choosing one parameter each time, keeping all the other parameters constant and running the model several times, each time for a different value of the parameter. Furthermore, for each parameter we tried several combination of the constant parameters to be sure we found the best values.
- Different model net structure: we tried some different structures and finally found that 4 layers of 512 dimension works best
- Model structure: we tried different structures like adding transformers to the model or changing activation function only to find that the basic structure works bets

3.2 Over fitting

One of our biggest concern was over fitting. As we saw in our tests most of the graphs saw constant improvement in train accuracy (as expected) but after some number of epochs test accuracy begun to decline, probably due to over fitting. To reduce this problem as much as possible we tried combining hyper-parameters in a lot of different combinations and choose the models that show improvements till the end. This may suggest that this models were able to learn the correct distribution (assuming the test data represents the true distribution)

4 Division of labor

Nitai:

- Writhing the model code and project structure
- Adding new models structures
- Thinking and implementing of new ideas
- Sending and analyse test runs

Daniel:

- Check that all parts of the model are working properly
- Thinking and implementing of new ideas
- Sending and analyse test runs
- Writhing the final report