Homework 3 Report **Ricardo Rei 78047**
Deep Structured Learning (IST, Fall 2018)
Prof. Andre Martins
TAs: Vlad Niculae and Erick Fonseca Due Date: 31/10/18

Question 1

Exercise 1

The pairwise features used in previous homework can exploit relations between adjacent pixels but convolutional layers go beyond that and can capture patterns between all the pixels in a given area. Another advantage of convolutional layers is that they do not require many parameters. In previous homework's, we have seen that feed-forward layers also learn to combine pixels in different ways in order to learn more powerful representations, but this kind of layers typically have a lot more parameters than convolutional layers, and for that reason, they are harder to train.

Exercise 2

Weight decay	Optimizer	LR	Minibatch
0.0	Adam	0.001	64

Table 1: Hyper-parameters used to train Exercise 1.2 CNN.

With the hyper-parameters presented in table 1, the model was able to achieve 95.07% accuracy in the train set, 90.47% accuracy in the dev set and 89.63% accuracy in the test set. Figure 1 presents the train and dev accuracy over training epochs, and figure 2 presents the train loss over the same epochs.

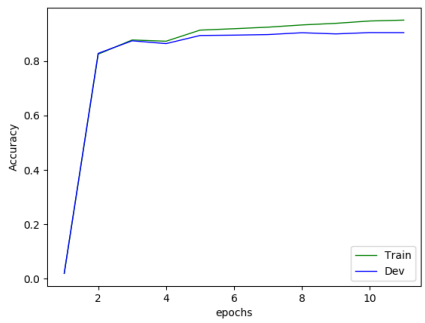

Figure 1: Train and Dev accuracy over training epochs.

Exercise 3

In theory, the filter weights of a well-trained convolution neural network should look smooth and without noisy patterns. This filters should also be more interpretable in lower layers and more abstract in higher layers, but from figure 3 and figure 4 it's hard to identify this kind of behavior. The filters presented in the mentioned figures look noisy and are hard to interpret.

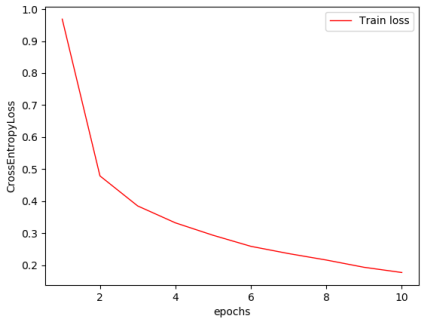

Figure 2: Cross Entropy Loss over training epochs.

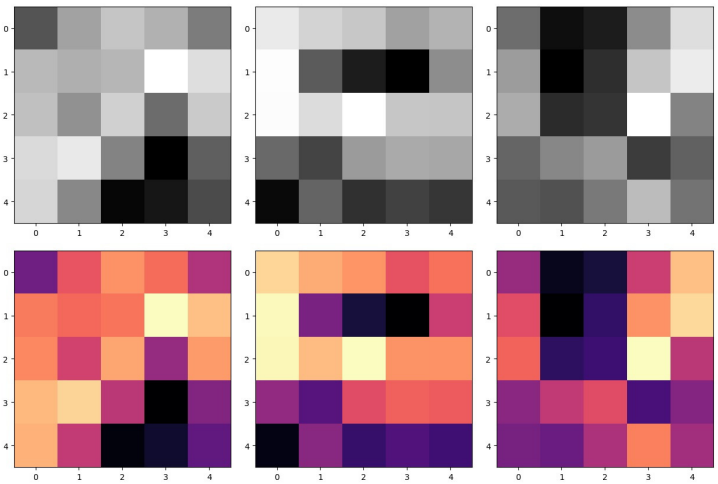

Figure 3: First layer filters in grey and magma color schemes.

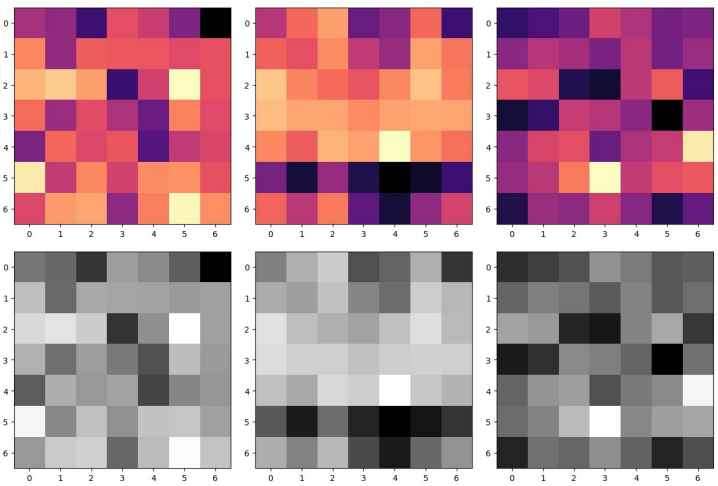

Figure 4: Second layer filters in grey and magma color schemes.

Exercise 4

For this exercise I developed 2 different models: ConvNN2 and ConvNN3.

ConvNN2 detailed description:

- A first convolutional layer with 20 channels and filters of size 5x5, stride of 1, and padding of 2.
- A relu activation applied to the end of the first conv. layer.
- Max pooling of size 2x2, with stride of 2.
- A second convolutional layer with 30 channels and filters of size 7x7, stride of 1, and padding of 3.
- A relu activation applied to the end of the second conv. layer.
- Max pooling of size 2x2, with stride of 2.
- An affine transformation to a space with 256 dimensions.
- A sigmoid activation applied to the resulting vector, followed by another affine transformation and an output softmax.

ConvNN3 detailed description:

- A first convolutional layer with 20 channels and filters of size 5x5, stride of 1, and padding of 2.
- A relu activation applied to the end of the first conv. layer.
- Max pooling of size 2x2, with stride of 2 and padding of 1.
- A second convolutional layer with 30 channels and filters of size 3x3, stride of 1, and padding of 2.
- A relu activation applied to the end of the second conv. layer.
- Max pooling of size 4x4, with stride of 2 and padding of 1.
- A third convolutional layer with 30 channels and filters of size 2x2, stride of 1, and padding of 1.
- A relu activation applied to the end of the thrid conv. layer.
- Max pooling of size 2x2, with stride of 2 and padding of 1.
- An affine transformation to a space with 256 dimensions.
- A sigmoid activation applied to the resulting vector, followed by another affine transformation and an output softmax.

Table 2 presents the train and dev accuracy of the presented models in comparison with the model of question 1 exercise 2 (our baseline). We can observe from table 2 that the presented models achieved similar performance as the baseline model and the small differences might be due to different random initialization.

Model	Train	Dev
Baseline	**95.07%**	90.47%
ConvNN2	94.14%	**90.79%**
ConvNN3	91.72%	89.74%

Table 2: Train and dev accuracy of the ConvNN2 and ConvNN3 models in comparison to our baseline.

Question 2

Exercise 1

For this exercise, I developed a bidirectional LSTM tagger that uses a feed-forward layer with 256 hidden units followed by 2 BiLSTM layers with the same hidden size. Table presents the training hyper-parameters. Figure 5 and figure 6 present the train/dev accuracy per epoch and the training loss per epoch, respectively. The described model achieved 99.85% train accuracy, 99.01% dev accuracy and 99.20% test accuracy.

Dropout	LR	Optimizer
0.1	0.001	Adam

(Note: the dropout is only applied between the LSTM hidden layers)

Table 3: Hyper-parameters used to train Exercise 2.1 LSTM.
(Note: the dropout is only applied between the LSTM hidden layers)

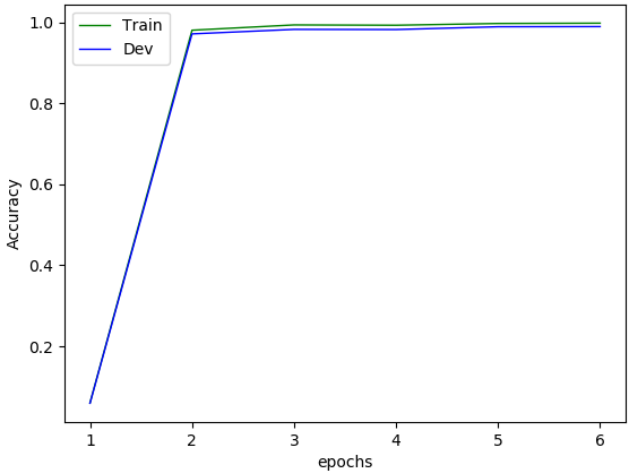

Figure 5: LSTM tagger Train and Dev accuracy for each train epoch.

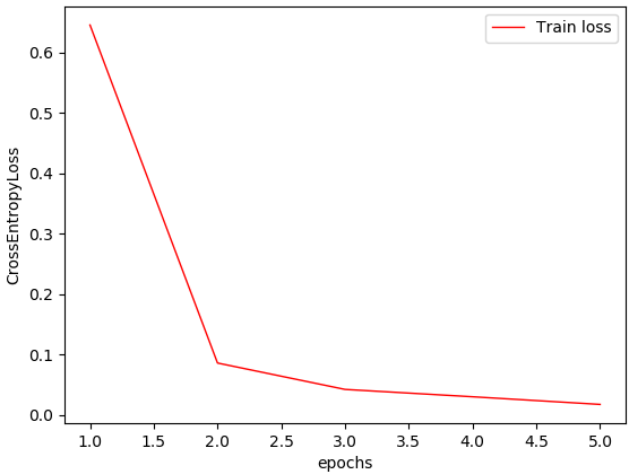

Figure 6: LSTM tagger cross Entropy Train loss for each train epoch.

Exercise 2

In a CRF we model the conditional probability of the output state sequence given an input sequence ($P(y_1, ..., y_n|x_1, ..., x_n)$). We do this by defining a feature map: $\Phi(x_1, ..., x_n, y_1, ..., y_n) \in R^d$ that maps an entire input sequence x paired with an entire state sequence y to some d-dimensional feature vector. Φ takes into account the probability of a label y_i given the entire input sequence ($P(y_i|x_1, ..., x_n)$) and the probability of consequent label pairs ($P(y_i, y_j|x_1, ..., x_n)$), in other words Φ is modelling the pairwise potentials and the unary potentials.

In a LSTM model, we are modelling the probability a label y_i conditioned on the input $x = (x_1, x_2, ..., x_n)$ as follows:

$$P(y_i|x_1, ..., x_n) = softmax(Wh_i + b)$$

If we look closely, we can observe that an LSTM is modelling non-linear unary potentials, and thus, we can use an LSTM to compute the unary potentials of a CRF model that until here were modelled as linear transformations of the input sequence.

Exercise 3

By replacing the feed-forward layer with a convolutional layer with 20 channels, a 5x5 filter, padding of 2 and stride of 1 I achieved 99.68% train accuracy, 98.42% dev accuracy and 98.37% test accuracy. Figure 7 and figure 8 present the train/dev accuracy per epoch and the training loss per epoch, respectively. All the training hyper-parameters were the same as the LSTM tagger presented in question 2 exercise 1 (table).

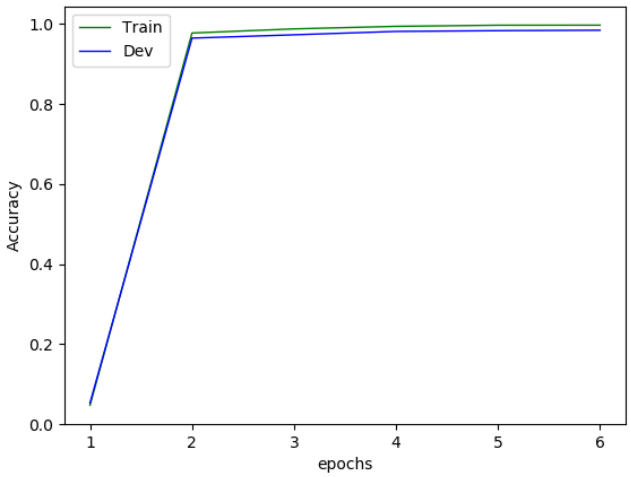

Figure 7: ConvLSTM tagger train and Dev accuracy for each train epoch.

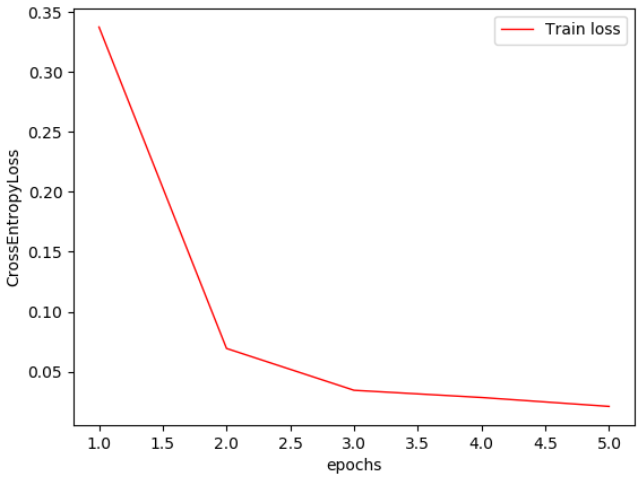

Figure 8: ConvLSTM tagger cross Entropy Train loss for each train epoch.