Exercise 4 – Speech Recognition Report

Eyal Orbach and Ofer Sabo

A short report on the model we have implemented in this exercise.

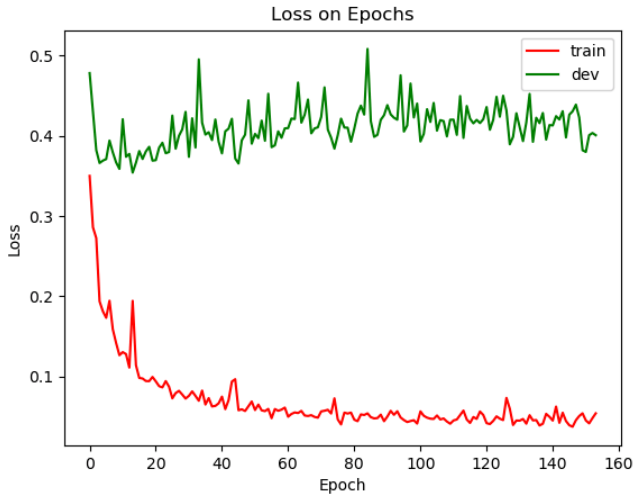
```
  def __init__(self):
      super(model_with_pooling, self).__init__()
      self.conv1 = nn.Conv2d(1, 15, 10, stride=1)
      nn.init.xavier_uniform_(self.conv1.weight)
      self.conv1_bn = nn.BatchNorm2d(15)

      self.pooling2d = nn.MaxPool2d(2,2)
      self.conv2 = nn.Conv2d(15, 1, 5,stride=1)
      nn.init.xavier_uniform_(self.conv2.weight)
      self.conv2_bn = nn.BatchNorm2d(1)

      self.dropout = nn.Dropout2d(p=0.4)
      self.rnn_module = nn.LSTM(dropout=0.2,input_size=72, hidden_size=128,
num_layers=2, batch_first=True, bidirectional=bidirectional)
      self.rnn2fc = nn.Linear(128*2, 128*2)
      self.h2o = nn.Linear(128*2, 27)
      self.softmax = nn.LogSoftmax(dim=2)
```

Above you may see the model code description. We will go over the layers one by one and later will discuss other parameters.

the first layer is a 2D convolutional filter, we decided to use this 2d filter since we know there is a strong connections between relatively close frequencies.

we have used a 15 channels output and kernel size of 15 because empirically they have produced the best result.
later we used a batch normalization.

as to the pooling layer, initially this layer wasn't part of the network we had, but when we analyzed the model loss, as can be seen in the graph which was generated in one of the first experiment we had, performance on the train and dev had a big gap.



Our conclusion was that we should insert a layer that will help us generalize better, for that we added the pooling layer. On top of that we also added a dropout factor in-between the LSTM layers.

Second convolutional layer would filter more complex features, second layer output is a single channel. This is helpful since a single channel is exactly what we want the RNN to get as an input. the result of these two convolutional layers is a sequence with length 42, each time frame has 72 features.

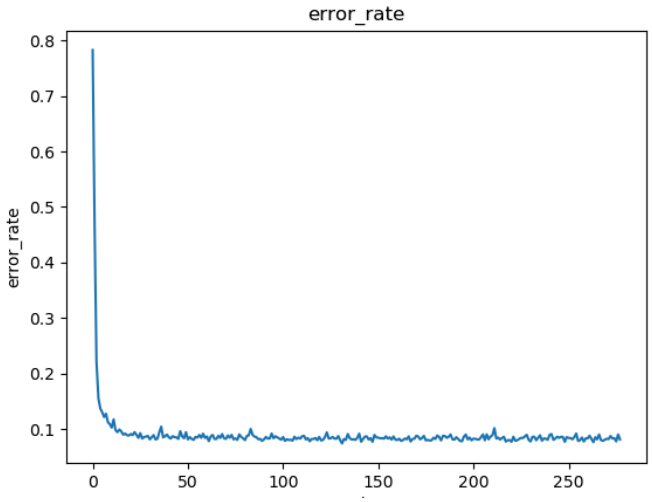
The RNN network we chose to implement is two layers BiLSTM. Due to the sequential input data we believe an RNN is a key factor in the model, this is essentially due to the context that the RNN can take into account.

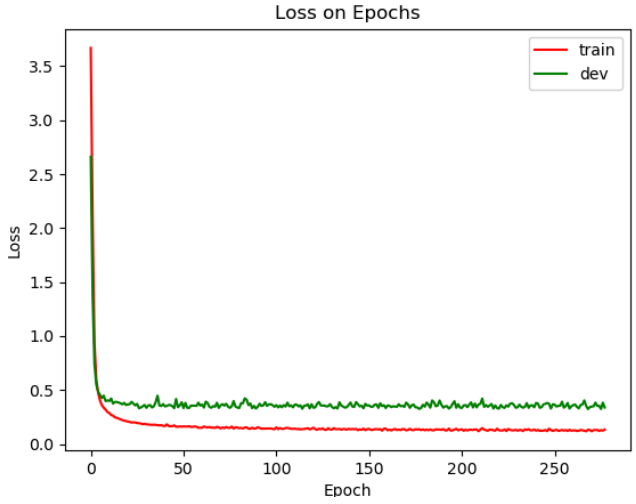
lastly the RNN output went into two fully connected layers which their output was the size if the number of classes. Here again we tried reducing to a single FC layer, but we saw that it produced poorer result. Difference in ER was 0.2%.

we transformed this score vector into a log probability with a log softmax layer.

Basically, we chose most of the hyper parameters after tweaking them and checking which one gives us the best result on the dev dataset.

Underneath you may see the graphs which our final model generated.


error_rate
error_rate
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250


Loss on Epochs
train
dev
Loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50
100
150
200
250
Epoch