Sentiment Analysis of Movie Reviews Using LSTM
NLP PROJECT REPORT: HXV180001

1) Successfully Installed the Environment and Executed the package : Yes

Downloaded the glove model from the link and successfully incorporated it in project, sample execution of project given below.

Output with default Config execution :

```
Namespace(batch_size=64, dpout_fc=0.0, dpout_model=0.0, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=64, n_classes=2, n_enc_layers=1,
n_epochs=20, nlipath='dataset/stsa/', nonlinear_fc=0, optimizer='adam', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234,
word_emb_dim=300, word_emb_path='dataset/GloVe/glove.840B.300d.txt')
** TRAIN DATA : Found 6920 pairs of train sentences.
** DEV DATA : Found 872 pairs of dev sentences.
** TEST DATA : Found 1821 pairs of test sentences.
Found 16517(/17576) words with glove vectors
Vocab size : 16517
NLINet(
  (encoder): LSTMEncoder(
    (enc_lstm): LSTM(300, 128)
  )
  (classifier): Sequential(
    (0): Linear(in_features=128, out_features=64, bias=True)
    (1): Linear(in_features=64, out_features=64, bias=True)
    (2): Linear(in_features=64, out_features=2, bias=True)
  )
)

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 84.0596
finalgrep : accuracy test : 84.6238
```

2) Have you made modifications on the hyperparameters? : Yes (Multiples)

1. Increasing the epoch to 40

Result :

No impact as no changes has been made to model structure also the data is less so model will not perform better than that even if the epoch is increased.

Namespace(batch_size=64, dpout_fc=0.0, dpout_model=0.0, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=64, n_classes=2, n_enc_layers=1, n_epochs=40, nlipath='dataset/stsa/', nonlinear_fc=0, optimizer='adam', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234, word_emb_dim=300, word_emb_path='dataset/GloVe/glove.840B.300d.txt')

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 84.0596
finalgrep : accuracy test : 84.6238

➔ nothing changes as such in terms of accuracy.

2. Increasing batch size to 128 with epoch : 20

```
NLINet(
  (encoder): LSTMEncoder(
    (enc_lstm): LSTM(300, 128)
  )
  (classifier): Sequential(
```

```
  (0): Linear(in_features=128, out_features=64, bias=True)
  (1): Linear(in_features=64, out_features=64, bias=True)
  (2): Linear(in_features=64, out_features=2, bias=True)
))
```

Result :

After 20 epochs the accuracy increases in test data from **84.6238** to **84.7337 , as the batch size increased so model will train faster on given data.**

TEST : Epoch 21
VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.6835
finalgrep : accuracy test : 84.7337

3. **Adding dropout in encoder, setting `--dpout_model = 1`**
   Result : Accuracy increases from previous 82% to 84.7%, adding dropout in classifier will make model neurons richer and comprehensive.

   VALIDATION : Epoch 1000000.0
   finalgrep : accuracy valid : 82.6835
   finalgrep : accuracy test : 84.7337

4. **Adding Dropout in classifier, `--dpout_fc = 1 and` `–dpout_model = 1, batch_size =128`**
   Result: No significant change

```
NLINet(
  (encoder): LSTMEncoder(
    (enc_lstm): LSTM(300, 128, dropout=1.0)
  )
  (classifier): Sequential(
    (0): Linear(in_features=128, out_features=64, bias=True)
    (1): Linear(in_features=64, out_features=64, bias=True)
    (2): Linear(in_features=64, out_features=2, bias=True)
  )
)
```

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.6835
finalgrep : accuracy test : 84.7337

5. **Adding non-linerality to be = 1**
   Result : Accuracy will boost a bit as model will have more layers to train but again the training time will also increase.

   VALIDATION : Epoch 1000000.0
   finalgrep : accuracy valid : 82.6835
   finalgrep : accuracy test : 84.7337

6. `--optimizer", type=str, default="adamax"`
   `Changing optimizer with learning rate of 0.8, standard gradient desent optimizer`

   Result: As learning rate was 0.8 the model will learn fast and gradient steps will be higher towards optimum, no such impact on accuracy as training data doesn't change also model architectural level remains same.

   VALIDATION : Epoch 1000000.0
   finalgrep : accuracy valid : 83.4862
   finalgrep : accuracy test : 83.3608

8. **Adding adam optimizer, --dpout model : 1 (Encoder Dropout) ,** `parser.add_argument("--optimizer", type=str, default="adam", help="adam or sgd,lr=0.1"):`

```
NLINet(
 (encoder): LSTMEncoder(
  (enc_lstm): LSTM(300, 128, dropout=0.4)
 )
 (classifier): Sequential(
  (0): Linear(in_features=128, out_features=64, bias=True)
  (1): Linear(in_features=64, out_features=64, bias=True)
  (2): Linear(in_features=64, out_features=2, bias=True)
 )
)
```

Result :
Changing to adam optimizer with batch-size = 128 and dropout in layers to 0.4 the model boost some accuracy and raises up to 84.7 from previous 82.6. The dropout helps to retain more and more neurons active in neural network.

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.6835
finalgrep : accuracy test : 84.7337
Increased accuracy from 84.6238 to **84.7337**

9. **`Increasing encoder batch size to 256,Changing encoder input dimensions.`**

```
parser.add_argument("--enc_lstm_dim", type=int, default=256, help="encoder nhid dimension")
```

`Result:`

No significant impact on accuracy as batch size will just increasing training data per iteration and model will converge soon.

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 83.4862
finalgrep : accuracy test : 83.5255

10. **`Reducing encoder batch size to 64,Changing encoder input dimensions.`**

```
parser.add_argument("--enc_lstm_dim", type=int, default=64, help="encoder nhid dimension")
```

`Result:`

No significant impact on accuracy as batch size will just reduce training data per iteration and model will converge later and need more epochs to converge.

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.5688
finalgrep : accuracy test : 83.9099

11. **Changing pool-type to mean :**

Result :
No such impact on accuracy as the training data is not huge.

```
parser.add_argument("--pool_type", type=str, default='mean', help="max or
mean")
```

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.6835
finalgrep : accuracy test : 84.7337

12. **Changing fc-dim to 128:**

```
parser.add_argument("--fc_dim", type=int, default=128, help="nhid of fc
layers")
```

Result :
No such impact on accuracy as it is not considered in model building.

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.2248
finalgrep : accuracy test : 82.9215

Sol. 3) Have you made structure level modifications of the model : Yes

Structural changes in Model.py class for modification and addition of nonlinear and linear layers while classification in neural network :

```
class NLINet(nn.Module):
    def __init__(self, config):
        super(NLINet, self).__init__()

        # classifier
        self.nonlinear_fc = config['nonlinear_fc']
        self.fc_dim = config['fc_dim']
        self.n_classes = config['n_classes']
        self.enc_lstm_dim = config['enc_lstm_dim']
        self.encoder_type = config['encoder_type']
        self.dpout_fc = config['dpout_fc']

        self.encoder = eval(self.encoder_type)(config)
        self.inputdim = self.enc_lstm_dim

        # self.classifier = nn.Sequential(
        #     nn.Linear( self.inputdim, self.fc_dim),
        #     nn.Linear(self.fc_dim, self.fc_dim),
        #     nn.Linear(self.fc_dim, self.n_classes)
        #     )

        if self.nonlinear_fc:
            self.classifier = nn.Sequential(
                nn.Dropout(p=self.dpout_fc),
                nn.Linear(self.inputdim, self.fc_dim),
                nn.Tanh(),
                nn.Dropout(p=self.dpout_fc),
                nn.Linear(self.fc_dim, self.fc_dim),
                nn.Tanh(),
                nn.Dropout(p=self.dpout_fc),
                nn.Linear(self.fc_dim, self.n_classes),
            )
        else:
            self.classifier = nn.Sequential(
                nn.Linear(self.inputdim, self.fc_dim),
                nn.Linear(self.fc_dim, self.fc_dim),
                nn.Linear(self.fc_dim, self.n_classes)
            )

    def forward(self, s1):
        # s1 : (s1, s1_len)
        u = self.encoder(s1)
        output = self.classifier(u)
        return output
```

Changes in Train.py class to update the batch-size and adding parameters for supporting and changing dimensions, dropout, pool type, fc-dim, optimizer and batch size parameters of model :

```
arser = argparse.ArgumentParser(description='NLI training')
# paths
arser.add_argument("--nlipath", type=str, default='dataset/stsa/', help="stsa data path ")
arser.add_argument("--outputdir", type=str, default='savedir/', help="Output directory")
arser.add_argument("--outputmodelname", type=str, default='model.pickle')
arser.add_argument("--word_emb_path", type=str, default="dataset/GloVe/glove.840B.300d.txt",help="word embedding file path")

# training
arser.add_argument("--n_epochs", type=int, default=40)
arser.add_argument("--batch_size", type=int, default=128)
arser.add_argument("--dpout_model", type=float, default=0.2, help="encoder dropout")
arser.add_argument("--dpout_fc", type=float, default=0.3, help="classifier dropout")
arser.add_argument("--nonlinear_fc", type=float, default=1.0, help="use nonlinearity in fc")
arser.add_argument("--optimizer", type=str, default="sgd,lr=0.7", help="adam or sgd,lr=0.1")

# model
arser.add_argument("--encoder_type", type=str, default='LSTMEncoder', help="see list of encoders")
arser.add_argument("--enc_lstm_dim", type=int, default=128, help="encoder nhid dimension")
arser.add_argument("--n_enc_layers", type=int, default=1, help="encoder num layers")
arser.add_argument("--fc_dim", type=int, default=128, help="nhid of fc layers")
arser.add_argument("--n_classes", type=int, default=2, help="positive/negative")
arser.add_argument("--pool_type", type=str, default='max', help="max or mean")

arser.add_argument("--seed", type=int, default=1234, help="seed")

# data
arser.add_argument("--word_emb_dim", type=int, default=300, help="word embedding dimension")
```

1) Changing the model by adding new non-linear layers and 30% dropout :

Result : Training time increases as model is non-linear and dropout and activation layers are added in classifier

Namespace(batch_size=64, dpout_fc=1, dpout_model=0.0, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=64, n_classes=2, n_enc_layers=1, n_epochs=50, nlipath='dataset/stsa/', nonlinear_fc=1, optimizer='adam', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234, word_emb_dim=300, word_emb_path='dataset/GloVe/glove.840B.300d.txt')
**** TRAIN DATA : Found 6920 pairs of train sentences.**
**** DEV DATA : Found 872 pairs of dev sentences.**
**** TEST DATA : Found 1821 pairs of test sentences.**
Found 16517(/17576) words with glove vectors
Vocab size : 16517
NLINet(
(encoder): LSTMEncoder(
(enc_lstm): LSTM(300, 128)
)
(classifier): Sequential(
(0): Dropout(p=1)
(1): Linear(in_features=128, out_features=64, bias=True)
(2): Tanh()
(3): Dropout(p=1)
(4): Linear(in_features=64, out_features=64, bias=True)
(5): Tanh()
(6): Dropout(p=1)
(7): Linear(in_features=64, out_features=2, bias=True)
)
)

TRAINING : Epoch 1
results : epoch 1 ; loss: 75.46; mean accuracy train : 52.1676

VALIDATION : Epoch 1
togrep : results : epoch 1 ; mean accuracy valid :50.9174
saving model at epoch 1

TEST : Epoch 51
VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 83.9174
finalgrep : accuracy test : 83.2176

2. Changing batch-size and reducing number of epochs with non linear data :

Result : Accuracy increases to 85%

Namespace(batch_size=128, dpout_fc=0.3, dpout_model=1.0, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=64, n_classes=2, n_enc_layers=1, n_epochs=30, nlipath='dataset/stsa/', nonlinear_fc=1.0, optimizer='adam', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234, word_emb_dim=300, word_emb_path='dataset/GloVe/glove.840B.300d.txt')
**** TRAIN DATA : Found 6920 pairs of train sentences.**
**** DEV DATA : Found 872 pairs of dev sentences.**
**** TEST DATA : Found 1821 pairs of test sentences.**
Found 16517(/17576) words with glove vectors
Vocab size : 16517
/Users/harshverma/anaconda3/lib/python3.6/site-packages/torch/nn/modules/rnn.py:46: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=1.0 and num_layers=1
"num_layers={}".format(dropout, num_layers))
NLINet(
(encoder): LSTMEncoder(
(enc_lstm): LSTM(300, 128, dropout=1.0)
)
(classifier): Sequential(
(0): Dropout(p=0.3)
(1): Linear(in_features=128, out_features=64, bias=True)
(2): Tanh()
(3): Dropout(p=0.3)
(4): Linear(in_features=64, out_features=64, bias=True)
(5): Tanh()
(6): Dropout(p=0.3)
(7): Linear(in_features=64, out_features=2, bias=True)
)
)

TRAINING : Epoch 1
results : epoch 1 ; loss: 31.69; mean accuracy train : 68.2225

accuracy valid :77.867

results : epoch 21 ; loss: 1.69; mean accuracy train : 99.0607

TEST : Epoch 31

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 84.0596
finalgrep : accuracy test : 85.0082

3. Increasing dropout and learning rate :

Namespace(batch_size=128, dpout_fc=0.3, dpout_model=0.2, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=128, n_classes=2, n_enc_layers=1, n_epochs=40, nlipath='dataset/stsa/', nonlinear_fc=1.0, optimizer='sgd,lr=0.7', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234, word_emb_dim=300, word_emb_path='dataset/GloVe/glove.840B.300d.txt')
** TRAIN DATA : Found 6920 pairs of train sentences.
** DEV DATA : Found 872 pairs of dev sentences.
** TEST DATA : Found 1821 pairs of test sentences.
Found 16517(/17576) words with glove vectors
Vocab size : 16517
/Users/harshverma/anaconda3/lib/python3.6/site-packages/torch/nn/modules/rnn.py:46: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.2 and num_layers=1
"num_layers={}".format(dropout, num_layers))
NLINet(
(encoder): LSTMEncoder(
(enc_lstm): LSTM(300, 128, dropout=0.2)
)
(classifier): Sequential(
(0): Dropout(p=0.3)
(1): Linear(in_features=128, out_features=128, bias=True)
(2): Tanh()
(3): Dropout(p=0.3)
(4): Linear(in_features=128, out_features=128, bias=True)
(5): Tanh()
(6): Dropout(p=0.3)
(7): Linear(in_features=128, out_features=2, bias=True)
)
)

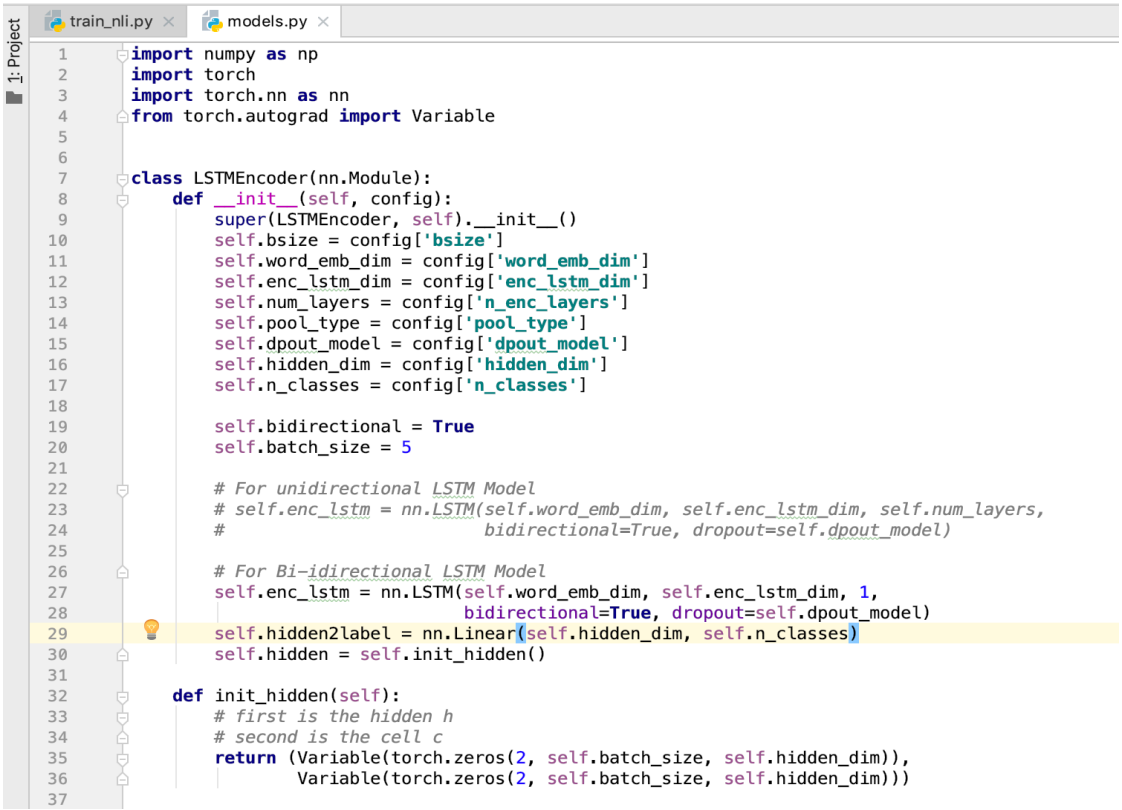
TRAINING : Epoch 1
results : epoch 1 ; loss: 38.09; mean accuracy train : 51.8064

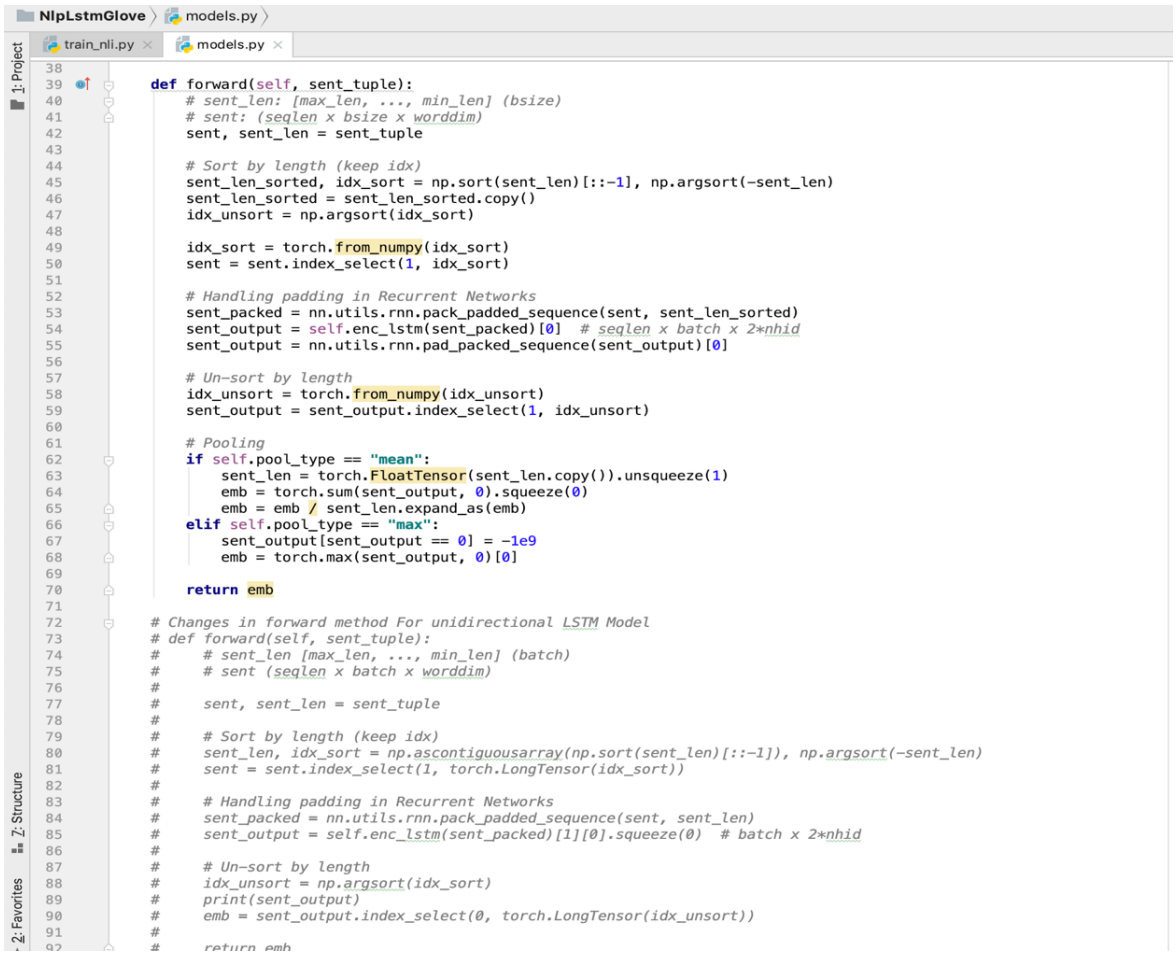
TEST : Epoch 41

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.4541
finalgrep : accuracy test : 83.3059

4) Have you extended the LSTM model to Bi-LSTM model : Yes

Attaching the screenshot of changes in code level (Model.py and train.py)

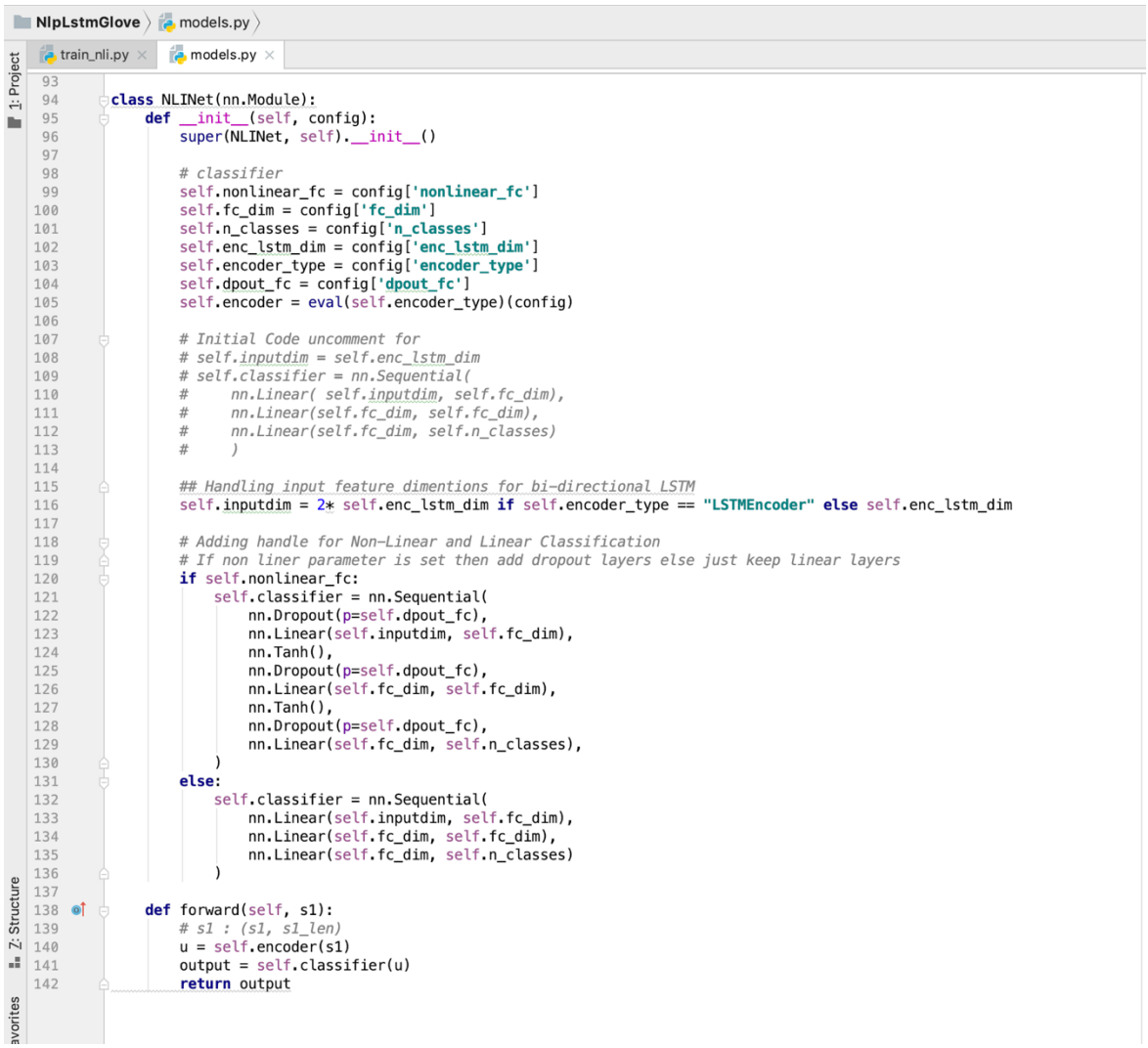





```python
    def forward(self, sent_tuple):
        # sent_len: [max_len, ..., min_len] (bsize)
        # sent: (seqlen x bsize x worddim)
        sent, sent_len = sent_tuple

        # Sort by length (keep idx)
        sent_len_sorted, idx_sort = np.sort(sent_len)[::-1], np.argsort(-sent_len)
        sent_len_sorted = sent_len_sorted.copy()
        idx_unsort = np.argsort(idx_sort)

        idx_sort = torch.from_numpy(idx_sort)
        sent = sent.index_select(1, idx_sort)

        # Handling padding in Recurrent Networks
        sent_packed = nn.utils.rnn.pack_padded_sequence(sent, sent_len_sorted)
        sent_output = self.enc_lstm(sent_packed)[0]  # seqlen x batch x 2*nhid
        sent_output = nn.utils.rnn.pad_packed_sequence(sent_output)[0]

        # Un-sort by length
        idx_unsort = torch.from_numpy(idx_unsort)
        sent_output = sent_output.index_select(1, idx_unsort)

        # Pooling
        if self.pool_type == "mean":
            sent_len = torch.FloatTensor(sent_len.copy()).unsqueeze(1)
            emb = torch.sum(sent_output, 0).squeeze(0)
            emb = emb / sent_len.expand_as(emb)
        elif self.pool_type == "max":
            sent_output[sent_output == 0] = -1e9
            emb = torch.max(sent_output, 0)[0]

        return emb

    # Changes in forward method For unidirectional LSTM Model
    # def forward(self, sent_tuple):
    #     # sent_len [max_len, ..., min_len] (batch)
    #     # sent (seqlen x batch x worddim)
    #
    #     sent, sent_len = sent_tuple
    #
    #     # Sort by length (keep idx)
    #     sent_len, idx_sort = np.ascontiguousarray(np.sort(sent_len)[::-1]), np.argsort(-sent_len)
    #     sent = sent.index_select(1, torch.LongTensor(idx_sort))
    #
    #     # Handling padding in Recurrent Networks
    #     sent_packed = nn.utils.rnn.pack_padded_sequence(sent, sent_len)
    #     sent_output = self.enc_lstm(sent_packed)[1][0].squeeze(0)  # batch x 2*nhid
    #
    #     # Un-sort by length
    #     idx_unsort = np.argsort(idx_sort)
    #     print(sent_output)
    #     emb = sent_output.index_select(0, torch.LongTensor(idx_unsort))
    #
    #     return emb
```

➔ **Changes in Neural Network to build a support for linear and non Linear Classifier with bi-directional LSTM encoder along with changes in Train.py for hidden layer dimensions.**

```python
parser = argparse.ArgumentParser(description='NLI training')
# paths
parser.add_argument("--nlipath", type=str, default='dataset/stsa/', help="stsa data path ")
parser.add_argument("--outputdir", type=str, default='savedir/', help="Output directory")
parser.add_argument("--outputmodelname", type=str, default='model.pickle')
parser.add_argument("--word_emb_path", type=str, default="dataset/GloVe/glove.840B.300d.txt",help="word embedding file path")

# training
parser.add_argument("--n_epochs", type=int, default=30)
parser.add_argument("--batch_size", type=int, default=128)
parser.add_argument("--dpout_model", type=float, default=0.2, help="encoder dropout")
parser.add_argument("--dpout_fc", type=float, default=0.3, help="classifier dropout")
parser.add_argument("--nonlinear_fc", type=float, default=1.0, help="use nonlinearity in fc")
parser.add_argument("--optimizer", type=str, default="adam", help="adam or sgd,lr=0.1")

# model
parser.add_argument("--encoder_type", type=str, default='LSTMEncoder', help="see list of encoders")
parser.add_argument("--enc_lstm_dim", type=int, default=128, help="encoder nhid dimension")
# Adding new Hidden Layer Dimentions Parameter
parser.add_argument("--hidden_dim", type=int, default=64, help="hidden dimension")
parser.add_argument("--n_enc_layers", type=int, default=1, help="encoder num layers")
parser.add_argument("--fc_dim", type=int, default=128, help="nhid of fc layers")
parser.add_argument("--n_classes", type=int, default=2, help="positive/negative")
parser.add_argument("--pool_type", type=str, default='max', help="max or mean")

parser.add_argument("--seed", type=int, default=1234, help="seed")

# data
parser.add_argument("--word_emb_dim", type=int, default=300, help="word embedding dimension")

params, _ = parser.parse_known_args()
```




```python
class NLINet(nn.Module):
    def __init__(self, config):
        super(NLINet, self).__init__()

        # classifier
        self.nonlinear_fc = config['nonlinear_fc']
        self.fc_dim = config['fc_dim']
        self.n_classes = config['n_classes']
        self.enc_lstm_dim = config['enc_lstm_dim']
        self.encoder_type = config['encoder_type']
        self.dpout_fc = config['dpout_fc']
        self.encoder = eval(self.encoder_type)(config)

        # Initial Code uncomment for
        # self.inputdim = self.enc_lstm_dim
        # self.classifier = nn.Sequential(
        #     nn.Linear( self.inputdim, self.fc_dim),
        #     nn.Linear(self.fc_dim, self.fc_dim),
        #     nn.Linear(self.fc_dim, self.n_classes)
        #     )

        ## Handling input feature dimentions for bi-directional LSTM
        self.inputdim = 2* self.enc_lstm_dim if self.encoder_type == "LSTMEncoder" else self.enc_lstm_dim

        # Adding handle for Non-Linear and Linear Classification
        # If non liner parameter is set then add dropout layers else just keep linear layers
        if self.nonlinear_fc:
            self.classifier = nn.Sequential(
                nn.Dropout(p=self.dpout_fc),
                nn.Linear(self.inputdim, self.fc_dim),
                nn.Tanh(),
                nn.Dropout(p=self.dpout_fc),
                nn.Linear(self.fc_dim, self.fc_dim),
                nn.Tanh(),
                nn.Dropout(p=self.dpout_fc),
                nn.Linear(self.fc_dim, self.n_classes),
            )
        else:
            self.classifier = nn.Sequential(
                nn.Linear(self.inputdim, self.fc_dim),
                nn.Linear(self.fc_dim, self.fc_dim),
                nn.Linear(self.fc_dim, self.n_classes)
                )

    def forward(self, s1):
        # s1 : (s1, s1_len)
        u = self.encoder(s1)
        output = self.classifier(u)
        return output
```

Changing to Linear Classification model with Bi-Directional LSTM:

Result:

Increased accuracy but indeed more time to train the bi-directional model, the model will increase accuracy more if more training data is provided to bi-directional model

Namespace(batch_size=128, dpout_fc=0.3, dpout_model=0.2, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=128, hidden_dim=128
n_classes=2, n_enc_layers=2, n_epochs=40, nlipath='dataset/stsa/', nonlinear_fc=0.0, optimizer='sgd,lr=0.7', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234, word_emb_dim=300,
word_emb_path='dataset/GloVe/glove.840B.300d.txt')
**** TRAIN DATA : Found 6920 pairs of train sentences.**
**** DEV DATA : Found 872 pairs of dev sentences.**
**** TEST DATA : Found 1821 pairs of test sentences.**
Found 16517(/17576) words with glove vectors
Vocab size : 16517
/Users/harshverma/anaconda3/lib/python3.6/site-packages/torch/nn/modules/rnn.py:46: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.2 and num_layers=1
"num_layers={}".format(dropout, num_layers))
NLINet(
(encoder): LSTMEncoder(
(enc_lstm): LSTM(300, 128, dropout=0.2, bidirectional=True)
)
(classifier): Sequential(
(0): Linear(in_features=256, out_features=128, bias=True)

(1): Linear(in_features=128, out_features=128, bias=True)
(2): Linear(in_features=128, out_features=2, bias=True)

TRAINING : Epoch 1
results : epoch 1 ; loss: 37.81; mean accuracy train : 54.6676

VALIDATION : Epoch 1
togrep : results : epoch 1 ; mean accuracy valid :73.5092
saving model at epoch 1

TEST : Epoch 41

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 82.9128
finalgrep : accuracy test : 84.2943

Changing to Non Linear Classification model with Bi-Directional LSTM:

Result:

Increased accuracy but indeed more time to train the bi-directional model, the model will increase accuracy more if more training data is provided to bi-directional model

Namespace(batch_size=128, dpout_fc=0.3, dpout_model=0.2, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=128, hidden_dim=64, n_classes=2, n_enc_layers=1, n_epochs=30, nlipath='dataset/stsa/', nonlinear_fc=1.0, optimizer='adam', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234, word_emb_dim=300, word_emb_path='dataset/GloVe/glove.840B.300d.txt')
**** TRAIN DATA : Found 6920 pairs of train sentences.**
**** DEV DATA : Found 872 pairs of dev sentences.**
**** TEST DATA : Found 1821 pairs of test sentences.**
Found 16517(/17576) words with glove vectors
Vocab size : 16517
/Users/harshverma/anaconda3/lib/python3.6/site-packages/torch/nn/modules/rnn.py:46: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.2 and num_layers=1
"num_layers={}".format(dropout, num_layers))
NLINet(
(encoder): LSTMEncoder(
(enc_lstm): LSTM(300, 128, dropout=0.2, bidirectional=True)
)
(classifier): Sequential(
(0): Dropout(p=0.3)
(1): Linear(in_features=256, out_features=128, bias=True)
(2): Tanh()
(3): Dropout(p=0.3)
(4): Linear(in_features=128, out_features=128, bias=True)
(5): Tanh()
(6): Dropout(p=0.3)
(7): Linear(in_features=128, out_features=2, bias=True)
)
)

Final Accuracy after 30 epochs :

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 83.6009
finalgrep : accuracy test : 85.777

Further Extension: Adding Hidden Layers to Bi-Directional LSTM Encoders :

Non-Linear model in classification with hidden layer in LSTM encoder.

Result:

Increased accuracy as hidden layers are added in encoder but indeed more time to train the bi-directional model, the model will increase accuracy more if more training data is provided to bi-directional model

Namespace(batch_size=128, dpout_fc=0.3, dpout_model=0.2, enc_lstm_dim=128, encoder_type='LSTMEncoder', fc_dim=128, hidden_dim=64, n_classes=2, n_enc_layers=1, n_epochs=30, nlipath='dataset/stsa/', nonlinear_fc=1.0, optimizer='adam', outputdir='savedir/', outputmodelname='model.pickle', pool_type='max', seed=1234, word_emb_dim=300, word_emb_path='dataset/GloVe/glove.840B.300d.txt')
**** TRAIN DATA : Found 6920 pairs of train sentences.**
**** DEV DATA : Found 872 pairs of dev sentences.**
**** TEST DATA : Found 1821 pairs of test sentences.**
Found 16517(/17576) words with glove vectors
Vocab size : 16517
/Users/harshverma/anaconda3/lib/python3.6/site-packages/torch/nn/modules/rnn.py:46: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.2 and num_layers=1
"num_layers={}".format(dropout, num_layers))
NLINet(
(encoder): LSTMEncoder(
(enc_lstm): LSTM(300, 128, dropout=0.2, bidirectional=True)
(hidden2label): Linear(in_features=64, out_features=2, bias=True)
)
(classifier): Sequential(
(0): Dropout(p=0.3)
(1): Linear(in_features=256, out_features=128, bias=True)
(2): Tanh()
(3): Dropout(p=0.3)
(4): Linear(in_features=128, out_features=128, bias=True)
(5): Tanh()
(6): Dropout(p=0.3)
(7): Linear(in_features=128, out_features=2, bias=True)
)
)

TRAINING : Epoch 1
results : epoch 1 ; loss: 29.58; mean accuracy train : 70.8382

VALIDATION : Epoch 1
togrep : results : epoch 1 ; mean accuracy valid :78.8991
saving model at epoch 1

TEST : Epoch 31

VALIDATION : Epoch 1000000.0
finalgrep : accuracy valid : 84.1743
finalgrep : accuracy test : 84.8435