After the GRU layer we will concatenate both, the average pooling and max pooling of the hidden representation and the last hidden state of GRU in order to prevent our model from forgetting infromations. This architecture is described in the following paper: https://arxiv.org/pdf/1801.06146.pdf. There is also the possibility to get rid of the last hidden state from our model at all, this kind of architecture, that uses max-pooling or avg-pooling is depicted in the paper: https://arxiv.org/pdf/1705.02364.pdf.

Building and training the model

Let's start with importing all indispensable libraries.

In [1]:
```
from batch_iterator import BatchIterator
from early_stopping import EarlyStopping
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch import device
from tqdm import tqdm_notebook
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from tensorboardX import SummaryWriter
```

Now, we are going to load the tarining and validation sets, but we will use only the clean_review column and label column.

In [2]:
```
# Import the dataset. Use clean_review and label columns
train_dataset = pd.read_csv('dataset/drugreview_feat_clean/train_feat_clean.c
                            usecols=['clean_review', 'rating'])

# Change columns order
train_dataset['label'] = train_dataset.rating >= 5
train_dataset = train_dataset[['clean_review', 'label']]
```

In [3]:
```
# Depict the first 5 rows of the training set
train_dataset = train_dataset.dropna()
train_dataset.head()
```

Out[3]:

	clean_review	label
2	young suffering severe extreme neck pain resul...	True
5	found work helping good nights sleep don'...	True
9	given medication gastroenterologist office wor...	False
12	recently laparoscopic hysterectomy know anesth...	True
13	mirena year experienced effects effects watch ...	False

In [4]:

```
# Import the dataset. Use clean_review and label columns
val_dataset = pd.read_csv('dataset/drugreview_feat_clean/val_feat_clean.csv',
                          usecols=['clean_review', 'rating'])

# Change columns order

val_dataset['label'] = val_dataset.rating >= 5
val_dataset = val_dataset[['clean_review', 'label']]
```

In [5]:

```
# Depict the first 5 rows of the validation set
val_dataset = val_dataset.dropna()
val_dataset.head()
```

Out[5]:

	clean_review	label
0	year old son took night went deep sea fishing ...	True
1	daughter epiduo grade junior year work wonders...	True
2	i've implant months day got totally felt ...	True
3	wanted wait days post couldn't results am...	True
4	colonoscopy best prep far morning took prep pm...	True

Below we will use the BatchIterator class defined in the previous notebook to create the vocabulary, trim sequences in terms of the rare word occurrence and the length, map words to their numerical representation (word2index), furthermore BatchIterator sorts dataset examples, generates batches, performs sequence padding and enables to use it instance to iterate through all batches.

In [6]:

```
train_iterator = BatchIterator(train_dataset, batch_size=256, vocab_created=F
                               word2index=None, sos_token='<SOS>', eos_token=
                               pad_token='<PAD>', min_word_count=3, max_vocab
                               use_pretrained_vectors=False, glove_path='glov
                               weights_file_name='glove/weights.npy')
```

```
Trimmed vocabulary using as minimum count threashold: count = 3.00
8674/21861 tokens has been retained
Trimmed input strings vocabulary
Trimmed input sequences lengths to the length of: 58
Mapped words to indices
Batches created
```

In [7]:
```
val_iterator = BatchIterator(val_dataset, batch_size=256, vocab_created=False
                             word2index=train_iterator.word2index, sos_token=
                             unk_token='<UNK>', pad_token='<PAD>', min_word_c
                             max_seq_len=0.9, use_pretrained_vectors=False, g
                             glove_name='glove.6B.100d.txt', weights_file_nam
```

```
Trimmed vocabulary using as minimum count threashold: count = 3.00
4655/11853 tokens has been retained
Trimmed input strings vocabulary
Trimmed input sequences lengths to the length of: 57
Mapped words to indices
Batches created
```

We have to check out how batches that we created look like before we pass them into the model. For the record, the set of batches for input and output variables is returned as a dictionary, thus we will just look at the dictionary keys to find out how to extract particular variables.

In [8]:
```
for batches in train_iterator:
    print(batches.keys())
    break
```

```
dict_keys(['input_seq', 'target', 'x_lengths'])
```

Notice that the output batch has the dimensions: (batch_size, seq_len)

In [9]:
```
for batches in train_iterator:
    # Unpack the dictionary of batches
    input_seq, target, x_lengths = batches['input_seq'], batches['target'], b
    print('input_seq shape: ', input_seq.size())
    print('target shape: ', target.size())
    print('x_lengths shape: ', x_lengths.size())
    break
```

```
input_seq shape:  torch.Size([256, 36])
target shape:  torch.Size([256])
x_lengths shape:  torch.Size([256])
```

In [10]:
```
for batches in val_iterator:
    # Unpack the dictionary of batches
    input_seq, target, x_lengths = batches['input_seq'], batches['target'], b
    print('input_seq shape: ', input_seq.size())
    print('target shape: ', target.size())
    print('x_lengths shape: ', x_lengths.size())
    break
```

```
input_seq shape:  torch.Size([256, 31])
target shape:  torch.Size([256])
x_lengths shape:  torch.Size([256])
```

Next step is to build the biGRU model.

In [11]:
```
class BiGRU(nn.Module):
    """BiDirectional GRU neural network model.

    Parameters
    ----------
    hidden_size: int
        Number of features in the hidden state.
    vocab_size: int
        The size of the vocabulary.
    embedding_dim: int
        The size of each embedding vector.
    output_size: int
        Number of classes.
    n_layers: int, optional (default=1)
        Number of stacked recurrent layers.
    dropout: float, optional (default=0.2)
        Probability of an element of the tensor to be zeroed.
    spatial_dropout: boolean, optional (default=True)
        Whether to use the spatial dropout.
    bidirectional: boolean, optional (default=True)
        Whether to use the bidirectional GRU.

    """

    def __init__(self, hidden_size, vocab_size, embedding_dim, output_size, n
                 spatial_dropout=True, bidirectional=True):

        # Inherit everything from the nn.Module
        super(BiGRU, self).__init__()

        # Initialize attributes
        self.hidden_size = hidden_size
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout_p = dropout
        self.spatial_dropout = spatial_dropout
```

```
        self.bidirectional = bidirectional
        self.n_directions = 2 if self.bidirectional else 1

        # Initialize layers
        self.embedding = nn.Embedding(self.vocab_size, self.embedding_dim)
        self.dropout = nn.Dropout(self.dropout_p)
        if self.spatial_dropout:
            self.spatial_dropout1d = nn.Dropout2d(self.dropout_p)
        self.gru = nn.GRU(self.embedding_dim, self.hidden_size, num_layers=se
                          dropout=(0 if n_layers == 1 else self.dropout_p), b
                          bidirectional=self.bidirectional)
        # Linear layer input size is equal to hidden_size * 3, becuase
        # we will concatenate max_pooling ,avg_pooling and last hidden state
        self.linear = nn.Linear(self.hidden_size * 3, self.output_size)


    def forward(self, input_seq, input_lengths, hidden=None):
        """Forward propagate through the neural network model.

        Parameters
        ----------
        input_seq: torch.Tensor
            Batch of input sequences.
        input_lengths: torch.LongTensor
            Batch containing sequences lengths.
        hidden: torch.FloatTensor, optional (default=None)
            Tensor containing initial hidden state.

        Returns
        -------
        torch.Tensor
            Logarithm of softmaxed input tensor.

        """
        # Extract batch_size
        self.batch_size = input_seq.size(0)

        # Embeddings shapes
        # Input: (batch_size,  seq_length)
        # Output: (batch_size, seq_length, embedding_dim)
        emb_out = self.embedding(input_seq)

        if self.spatial_dropout:
            # Convert to (batch_size, embedding_dim, seq_length)
            emb_out = emb_out.permute(0, 2, 1)
            emb_out = self.spatial_dropout1d(emb_out)
            # Convert back to (batch_size, seq_length, embedding_dim)
            emb_out = emb_out.permute(0, 2, 1)
        else:
            emb_out = self.dropout(emb_out)

        # Pack padded batch of sequences for RNN module
        packed_emb = nn.utils.rnn.pack_padded_sequence(emb_out, input_lengths
```

```
        # GRU input/output shapes, if batch_first=True
        # Input: (batch_size, seq_len, embedding_dim)
        # Output: (batch_size, seq_len, hidden_size*num_directions)
        # Number of directions = 2 when used bidirectional, otherwise 1
        # shape of hidden: (n_layers x num_directions, batch_size, hidden_siz
        # Hidden state defaults to zero if not provided
        gru_out, hidden = self.gru(packed_emb, hidden)
        # gru_out: tensor containing the output features h_t from the last la
        # gru_out comprises all the hidden states in the last layer ("last" d
        # For biGRu gru_out is the concatenation of a forward GRU representat
        # hidden (h_n) comprises the hidden states after the last timestep

        # Extract and sum last hidden state
        # Input hidden shape: (n_layers x num_directions, batch_size, hidden_
        # Separate hidden state layers
        hidden = hidden.view(self.n_layers, self.n_directions, self.batch_siz
        last_hidden = hidden[-1]
        # last hidden shape (num_directions, batch_size, hidden_size)
        # Sum the last hidden state of forward and backward layer
        last_hidden = torch.sum(last_hidden, dim=0)
        # Summed last hidden shape (batch_size, hidden_size)

        # Pad a packed batch
        # gru_out output shape: (batch_size, seq_len, hidden_size*num_directi
        gru_out, lengths = nn.utils.rnn.pad_packed_sequence(gru_out, batch_fi

        # Sum the gru_out along the num_directions
        if self.bidirectional:
            gru_out = gru_out[:,:,:self.hidden_size] + gru_out[:,:,self.hidde

        # Select the maximum value over each dimension of the hidden represen
        # Permute the input tensor to dimensions: (batch_size, hidden, seq_le
        # Output dimensions: (batch_size, hidden_size)
        max_pool = F.adaptive_max_pool1d(gru_out.permute(0,2,1), (1,)).view(s

        # Consider the average of the representations (mean pooling)
        # Sum along the batch axis and divide by the corresponding lengths (F
        # Output shape: (batch_size, hidden_size)
        avg_pool = torch.sum(gru_out, dim=1) / lengths.view(-1,1).type(torch.

        # Concatenate max_pooling, avg_pooling and last hidden state tensors
        concat_out = torch.cat([last_hidden, max_pool, avg_pool], dim=1)

        #concat_out = self.dropout(concat_out)
        out = self.linear(concat_out)
        return F.log_softmax(out, dim=-1)


    def add_loss_fn(self, loss_fn):
        """Add loss function to the model.

        """
```

```
        self.loss_fn = loss_fn


    def add_optimizer(self, optimizer):
        """Add optimizer to the model.

        """
        self.optimizer = optimizer


    def add_device(self, device=torch.device('cpu')):
        """Specify the device.

        """
        self.device = device


    def train_model(self, train_iterator):
        """Perform single training epoch.

        Parameters
        ----------
        train_iterator: BatchIterator
            BatchIterator class object containing training batches.

        Returns
        -------
        train_losses: list
            List of the training average batch losses.
        avg_loss: float
            Average loss on the entire training set.
        accuracy: float
            Models accuracy on the entire training set.

        """
        self.train()

        train_losses = []
        losses = []
        losses_list = []
        num_seq = 0
        batch_correct = 0

        for i, batches in tqdm_notebook(enumerate(train_iterator, 1), total=l
            input_seq, target, x_lengths = batches['input_seq'], batches['tar

            input_seq.to(self.device)
            target.to(self.device)
            x_lengths.to(self.device)

            self.optimizer.zero_grad()

            pred = self.forward(input_seq, x_lengths)
```

```
            loss = self.loss_fn(pred, target)
            loss.backward()
            losses.append(loss.data.cpu().numpy())
            self.optimizer.step()

            losses_list.append(loss.data.cpu().numpy())

            pred = torch.argmax(pred, 1)

            if self.device.type == 'cpu':
                batch_correct += (pred.cpu() == target.cpu()).sum().item()

            else:
                batch_correct += (pred == target).sum().item()

            num_seq += len(input_seq)

            if i % 100 == 0:
                avg_train_loss = np.mean(losses)
                train_losses.append(avg_train_loss)

                accuracy = batch_correct / num_seq

                print('Iteration: {}. Average training loss: {:.4f}. Accuracy
                      .format(i, avg_train_loss, accuracy))

                losses = []

            avg_loss = np.mean(losses_list)
            accuracy = batch_correct / num_seq

        return train_losses, avg_loss, accuracy


    def evaluate_model(self, eval_iterator, conf_mtx=False):
        """Perform the one evaluation epoch.

        Parameters
        ----------
        eval_iterator: BatchIterator
            BatchIterator class object containing evaluation batches.
        conf_mtx: boolean, optional (default=False)
            Whether to print the confusion matrix at each epoch.

        Returns
        -------
        eval_losses: list
            List of the evaluation average batch losses.
        avg_loss: float
            Average loss on the entire evaluation set.
        accuracy: float
            Models accuracy on the entire evaluation set.
        conf_matrix: list
```

```
            Confusion matrix.

        """
        self.eval()

        eval_losses = []
        losses = []
        losses_list = []
        num_seq = 0
        batch_correct = 0
        pred_total = torch.LongTensor()
        target_total = torch.LongTensor()

        with torch.no_grad():
            for i, batches in tqdm_notebook(enumerate(eval_iterator, 1), tota
                input_seq, target, x_lengths = batches['input_seq'], batches[

                input_seq.to(self.device)
                target.to(self.device)
                x_lengths.to(self.device)

                pred = self.forward(input_seq, x_lengths)
                loss = self.loss_fn(pred, target)
                losses.append(loss.data.cpu().numpy())
                losses_list.append(loss.data.cpu().numpy())

                pred = torch.argmax(pred, 1)

                if self.device.type == 'cpu':
                    batch_correct += (pred.cpu() == target.cpu()).sum().item(

                else:
                    batch_correct += (pred == target).sum().item()

                num_seq += len(input_seq)

                pred_total = torch.cat([pred_total, pred], dim=0)
                target_total = torch.cat([target_total, target], dim=0)

                if i % 100 == 0:
                    avg_batch_eval_loss = np.mean(losses)
                    eval_losses.append(avg_batch_eval_loss)

                    accuracy = batch_correct / num_seq

                    print('Iteration: {}. Average evaluation loss: {:.4f}. Ac
                          .format(i, avg_batch_eval_loss, accuracy))

                    losses = []

            avg_loss_list = []

            avg_loss = np.mean(losses_list)
```

```
            accuracy = batch_correct / num_seq

            conf_matrix = confusion_matrix(target_total.view(-1), pred_total.

        if conf_mtx:
            print('\tConfusion matrix: ', conf_matrix)

        return eval_losses, avg_loss, accuracy, conf_matrix
```

Now we will instantiate the model, add loss function, optimizer, and device to it and begin the training.

In [23]:

```
# Initialize parameters
hidden_size = 8
vocab_size = len(train_iterator.word2index)
embedding_dim = 200
output_size = 2
n_layers = 1
dropout = 0.5
learning_rate = 0.001
epochs = 20
spatial_dropout = True

# Check whether system supports CUDA
CUDA = torch.cuda.is_available()

model = BiGRU(hidden_size, vocab_size, embedding_dim, output_size, n_layers,
              spatial_dropout, bidirectional=True)

# Move the model to GPU if possible
if CUDA:
    model.cuda()

model.add_loss_fn(nn.NLLLoss())

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
model.add_optimizer(optimizer)

device = torch.device('cuda' if CUDA else 'cpu')

model.add_device(device)

# Instantiate the EarlyStopping
early_stop = EarlyStopping(wait_epochs=1)

train_losses_list, train_avg_loss_list, train_accuracy_list = [], [], []
eval_avg_loss_list, eval_accuracy_list, conf_matrix_list = [], [], []

for epoch in range(epochs):

    print('\nStart epoch [{}/{}]'.format(epoch+1, epochs))
```

```
    train_losses, train_avg_loss, train_accuracy = model.train_model(train_it

    train_losses_list.append(train_losses)
    train_avg_loss_list.append(train_avg_loss)
    train_accuracy_list.append(train_accuracy)

    _, eval_avg_loss, eval_accuracy, conf_matrix = model.evaluate_model(val_i

    eval_avg_loss_list.append(eval_avg_loss)
    eval_accuracy_list.append(eval_accuracy)
    conf_matrix_list.append(conf_matrix)

    print('\nEpoch [{}/{}]: Train accuracy: {:.3f}. Train loss: {:.4f}. Evalu
          .format(epoch+1, epochs, train_accuracy, train_avg_loss, eval_accur

    if early_stop.stop(eval_avg_loss, model, delta=0.003):
        break
```

```
Start epoch [1/20]


Epoch [1/20]: Train accuracy: 0.675. Train loss: 0.6227. Evaluation accuracy:
0.757. Evaluation loss: 0.5564

Start epoch [2/20]


Epoch [2/20]: Train accuracy: 0.747. Train loss: 0.5696. Evaluation accuracy:
0.757. Evaluation loss: 0.5481

Start epoch [3/20]


Epoch [3/20]: Train accuracy: 0.747. Train loss: 0.5650. Evaluation accuracy:
0.757. Evaluation loss: 0.5423

Start epoch [4/20]


Epoch [4/20]: Train accuracy: 0.748. Train loss: 0.5588. Evaluation accuracy:
0.757. Evaluation loss: 0.5362

Start epoch [5/20]


Epoch [5/20]: Train accuracy: 0.748. Train loss: 0.5520. Evaluation accuracy:
0.757. Evaluation loss: 0.5283

Start epoch [6/20]


Epoch [6/20]: Train accuracy: 0.748. Train loss: 0.5467. Evaluation accuracy:
0.758. Evaluation loss: 0.5184

Start epoch [7/20]
```

```
Epoch [7/20]: Train accuracy: 0.749. Train loss: 0.5382. Evaluation accuracy:
0.761. Evaluation loss: 0.5074

Start epoch [8/20]

Epoch [8/20]: Train accuracy: 0.751. Train loss: 0.5249. Evaluation accuracy:
0.763. Evaluation loss: 0.4938

Start epoch [9/20]

Epoch [9/20]: Train accuracy: 0.756. Train loss: 0.5122. Evaluation accuracy:
0.773. Evaluation loss: 0.4799

Start epoch [10/20]

Epoch [10/20]: Train accuracy: 0.762. Train loss: 0.5000. Evaluation accuracy:
0.779. Evaluation loss: 0.4675

Start epoch [11/20]

Epoch [11/20]: Train accuracy: 0.772. Train loss: 0.4855. Evaluation accuracy:
0.788. Evaluation loss: 0.4560

Start epoch [12/20]

Epoch [12/20]: Train accuracy: 0.780. Train loss: 0.4688. Evaluation accuracy:
0.795. Evaluation loss: 0.4491

Start epoch [13/20]

Epoch [13/20]: Train accuracy: 0.787. Train loss: 0.4546. Evaluation accuracy:
0.799. Evaluation loss: 0.4436

Start epoch [14/20]

Epoch [14/20]: Train accuracy: 0.795. Train loss: 0.4422. Evaluation accuracy:
0.803. Evaluation loss: 0.4386

Start epoch [15/20]

Epoch [15/20]: Train accuracy: 0.799. Train loss: 0.4339. Evaluation accuracy:
0.803. Evaluation loss: 0.4362

Start epoch [16/20]

Epoch [16/20]: Train accuracy: 0.808. Train loss: 0.4240. Evaluation accuracy:
0.805. Evaluation loss: 0.4346

Start epoch [17/20]
```

```
Epoch [17/20]: Train accuracy: 0.812. Train loss: 0.4155. Evaluation accuracy:
0.808. Evaluation loss: 0.4332

Start epoch [18/20]


Epoch [18/20]: Train accuracy: 0.813. Train loss: 0.4064. Evaluation accuracy:
0.811. Evaluation loss: 0.4332

Start epoch [19/20]


Epoch [19/20]: Train accuracy: 0.824. Train loss: 0.3958. Evaluation accuracy:
0.811. Evaluation loss: 0.4339

Start epoch [20/20]


Epoch [20/20]: Train accuracy: 0.822. Train loss: 0.3902. Evaluation accuracy:
0.813. Evaluation loss: 0.4340
```

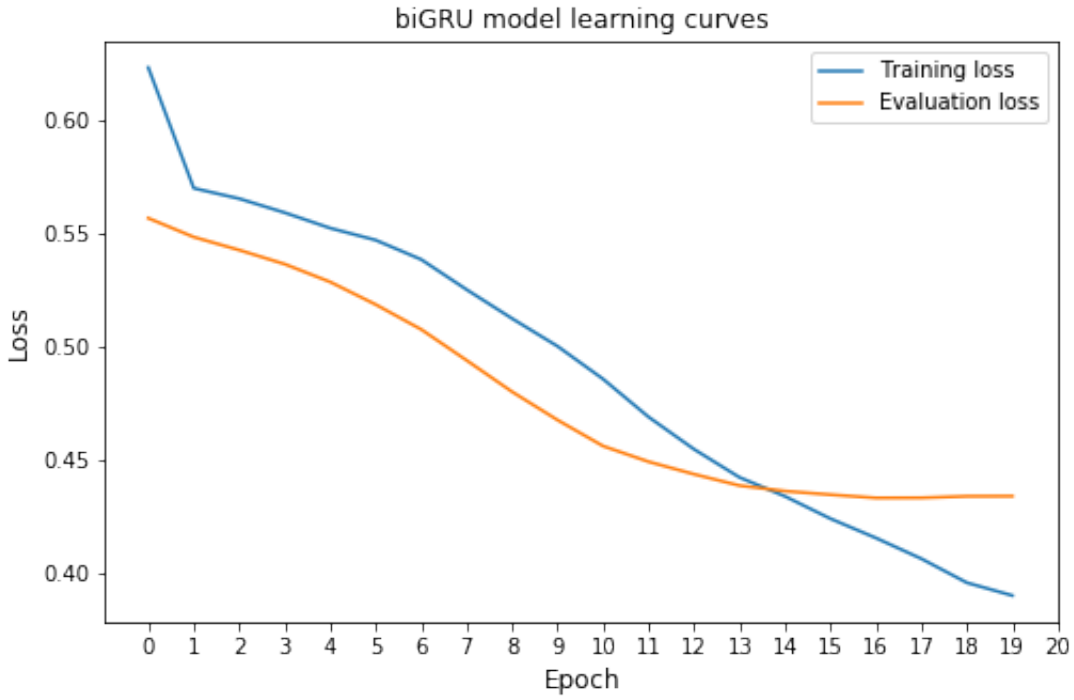
In [20]:

```
# Add the dataset initial loss
#print(len(train_losses_list))
#train_avg_loss_list.insert(0, train_losses_list[0])
#eval_avg_loss_list.insert(0, train_losses_list[0])
```

```
20
```

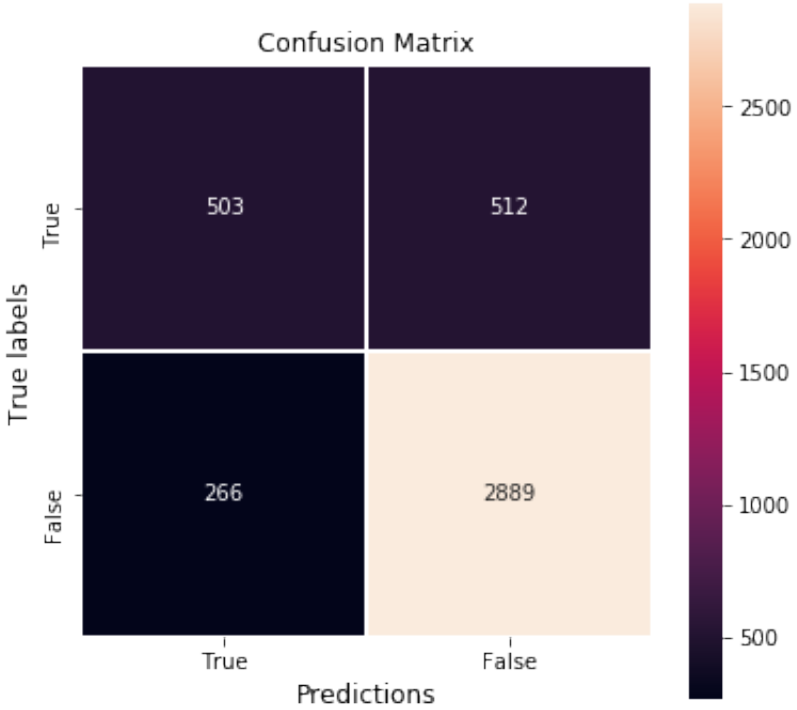
In [24]:

```
# Plot the training and the validation learning curve
plt.figure(figsize=(8,5))
plt.plot(train_avg_loss_list, label='Training loss')
plt.plot(eval_avg_loss_list, label='Evaluation loss')
plt.xlabel('Epoch', size=12)
plt.ylabel('Loss', size=12)
plt.title('biGRU model learning curves')
plt.xticks(ticks=range(21))
plt.legend()
plt.show()
```



In [25]:

```
# Confusion matrix
plt.figure(figsize=(6,6))
ax = sns.heatmap(conf_matrix, fmt='d', annot=True, linewidths=1, square=True)
ax.set_xlabel('Predictions', size=12)
ax.set_ylabel('True labels', size=12)
ax.set_title('Confusion Matrix', size=12);
ax.xaxis.set_ticklabels(['True', 'False'])
ax.yaxis.set_ticklabels(['True', 'False'])
ax.set_ylim(2,0)
plt.show()
```



The model achieved the validation accuracy of 0.878, while the training accuracy was 0.908. The model's best state was saved to the *checkpoint.pt* file in the current directory. The training wasn't stopped by EarlyStopping object because the validation loss changes were too small and fluctuated near the same value.

The training process that is presented above regards the model with the tuned hyperparameters. The steps we went through when doing hyperparameters fine-tuning are listed in the next section.

The summary - final set of hyperparameters

Considering all above training trials, we can draw the following conclusions:

- increasing the dropout probability helps in reducing the model's overfitting.
- spatial dropout works better in terms of decreasing the variance problem than the traditional dropout.
- the most improvement in reducing overfitting is due to the reduction of hidden_size.
- using stacked GRU doesn't improve in our case the model's performance.
- reducing the batch_size doesn't significantly affect the model's learning ability what is rather unexpected while increasing the batch_size does improve a bit the model's performance.

The following are the hyperparameters that will be used to finaly train our neural network:

- hidden_size = 8
- embedding_dim = 200
- n_layers = 1
- dropout = 0.5
- learning_rate = 0.001
- epochs = 20
- spatial_dropout = True
- batch_size = 256
- min_word_count = 3
- max_seq_len = 0.9

Below we will use the *tensorboardX* to create the graph of our neural network model that has been depicted at the top of this notebook. You can encounter torch._C.Value issue while using *add.graph()* method, to tackle that I recommend following the *github* thread devoted to this topic:
https://github.com/lanpa/tensorboardX/issues/483
namely, you can try to build tensorboardX from source with:
git clone https://github.com/lanpa/tensorboardX && cd tensorboardX && python setup.py install

In [26]:

```
hidden_size = 4
vocab_size = len(train_iterator.word2index)
embedding_dim = 200
n_layers = 2
output_size = 2
spatial_dropout = True
dropout = 0.5

writer = SummaryWriter('runs/exp-1')

for batch in train_iterator:
    input_seq, _, x_lengths = batch['input_seq'], batch['target'], batch['x_l

with SummaryWriter(comment='Model graph') as w:
    w.add_graph(BiGRU(hidden_size, vocab_size, embedding_dim, output_size, n_
                      spatial_dropout, bidirectional=True), (input_seq, x_len
```

```
graph(%self : ClassType<BiGRU>,
      %input_seq : Long(256, 28),
      %lengths.1 : Long(256)):
  %1 : ClassType<Embedding> = prim::GetAttr[name="embedding"](%self)
  %weight.1 : Tensor = prim::GetAttr[name="weight"](%1)
  %5 : ClassType<GRU> = prim::GetAttr[name="gru"](%self)
  %6 : Tensor = prim::GetAttr[name="weight_ih_l0"](%5)
  %7 : Tensor = prim::GetAttr[name="weight_hh_l0"](%5)
  %8 : Tensor = prim::GetAttr[name="bias_ih_l0"](%5)
  %9 : Tensor = prim::GetAttr[name="bias_hh_l0"](%5)
  %10 : Tensor = prim::GetAttr[name="weight_ih_l0_reverse"](%5)
  %11 : Tensor = prim::GetAttr[name="weight_hh_l0_reverse"](%5)
  %12 : Tensor = prim::GetAttr[name="bias_ih_l0_reverse"](%5)
  %13 : Tensor = prim::GetAttr[name="bias_hh_l0_reverse"](%5)
  %14 : Tensor = prim::GetAttr[name="weight_ih_l1"](%5)
  %15 : Tensor = prim::GetAttr[name="weight_hh_l1"](%5)
  %16 : Tensor = prim::GetAttr[name="bias_ih_l1"](%5)
  %17 : Tensor = prim::GetAttr[name="bias_hh_l1"](%5)
  %18 : Tensor = prim::GetAttr[name="weight_ih_l1_reverse"](%5)
  %19 : Tensor = prim::GetAttr[name="weight_hh_l1_reverse"](%5)
  %20 : Tensor = prim::GetAttr[name="bias_ih_l1_reverse"](%5)
  %21 : Tensor = prim::GetAttr[name="bias_hh_l1_reverse"](%5)
  %22 : ClassType<Linear> = prim::GetAttr[name="linear"](%self)
  %weight : Tensor = prim::GetAttr[name="weight"](%22)
  %bias : Tensor = prim::GetAttr[name="bias"](%22)
  %27 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:74:0
  %28 : int = aten::size(%input_seq, %27), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:74:0
  %29 : Long() = prim::NumToTensor(%28), scope: BiGRU
  %161 : int = aten::Int(%29), scope: BiGRU
  %98 : int = aten::Int(%29), scope: BiGRU
  %30 : int = prim::Constant[value=-1](), scope: BiGRU/Embedding[embedding] #
/Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/f
unctional.py:1467:0
  %31 : bool = prim::Constant[value=0](), scope: BiGRU/Embedding[embedding] #
/Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/f
```

```
unctional.py:1467:0
  %32 : bool = prim::Constant[value=0](), scope: BiGRU/Embedding[embedding] #
/Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/f
unctional.py:1467:0
  %emb_out.1 : Float(256, 28, 200) = aten::embedding(%weight.1, %input_seq, %3
0, %31, %32), scope: BiGRU/Embedding[embedding] # /Users/vsitham/opt/anaconda3
/envs/py3.7/lib/python3.7/site-packages/torch/nn/functional.py:1467:0
  %34 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:83:0
  %35 : int = prim::Constant[value=2](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:83:0
  %36 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:83:0
  %37 : int[] = prim::ListConstruct(%34, %35, %36), scope: BiGRU
  %input.1 : Float(256!, 200!, 28!) = aten::permute(%emb_out.1, %37), scope: B
iGRU # <ipython-input-11-5f3ae3dfaec2>:83:0
  %39 : float = prim::Constant[value=0.5](), scope: BiGRU/Dropout2d[spatial_dr
opout1d] # /Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages
/torch/nn/functional.py:844:0
  %40 : bool = prim::Constant[value=0](), scope: BiGRU/Dropout2d[spatial_dropo
ut1d] # /Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/to
rch/nn/functional.py:844:0
  %emb_out : Float(256!, 200!, 28!) = aten::feature_dropout(%input.1, %39, %40
), scope: BiGRU/Dropout2d[spatial_dropout1d] # /Users/vsitham/opt/anaconda3/en
vs/py3.7/lib/python3.7/site-packages/torch/nn/functional.py:844:0
  %42 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:86:0
  %43 : int = prim::Constant[value=2](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:86:0
  %44 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:86:0
  %45 : int[] = prim::ListConstruct(%42, %43, %44), scope: BiGRU
  %input.2 : Float(256, 28, 200) = aten::permute(%emb_out, %45), scope: BiGRU
# <ipython-input-11-5f3ae3dfaec2>:86:0
  %47 : Device = prim::Constant[value="cpu"](), scope: BiGRU # /Users/vsitham/
opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:265
:0
  %48 : int = prim::Constant[value=4](), scope: BiGRU # /Users/vsitham/opt/ana
conda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:265:0
  %49 : bool = prim::Constant[value=0](), scope: BiGRU # /Users/vsitham/opt/an
aconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:265:0
  %50 : bool = prim::Constant[value=0](), scope: BiGRU # /Users/vsitham/opt/an
aconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:265:0
  %lengths.2 : Long(256) = aten::to(%lengths.1, %47, %48, %49, %50), scope: Bi
GRU # /Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torc
h/nn/utils/rnn.py:265:0
  %52 : bool = prim::Constant[value=1](), scope: BiGRU # /Users/vsitham/opt/an
aconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:275:0
  %input.3 : Float(6948, 200), %batch_sizes : Long(28) = aten::_pack_padded_se
quence(%input.2, %lengths.2, %52), scope: BiGRU # /Users/vsitham/opt/anaconda3
/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:275:0
  %58 : int = prim::Constant[value=4](), scope: BiGRU/GRU[gru] # /Users/vsitha
m/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.py
:691:0
  %59 : int = prim::Constant[value=256](), scope: BiGRU/GRU[gru] # /Users/vsit
ham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.
py:691:0
```

```
  %60 : int = prim::Constant[value=4](), scope: BiGRU/GRU[gru] # /Users/vsitha
m/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.py
:691:0
  %61 : int[] = prim::ListConstruct(%58, %59, %60), scope: BiGRU/GRU[gru]
  %62 : int = prim::Constant[value=6](), scope: BiGRU/GRU[gru] # /Users/vsitha
m/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.py
:691:0
  %63 : int = prim::Constant[value=0](), scope: BiGRU/GRU[gru] # /Users/vsitha
m/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.py
:691:0
  %64 : Device = prim::Constant[value="cpu"](), scope: BiGRU/GRU[gru] # /Users
/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules
/rnn.py:691:0
  %65 : bool = prim::Constant[value=0](), scope: BiGRU/GRU[gru] # /Users/vsith
am/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.p
y:691:0
  %hx : Float(4, 256, 4) = aten::zeros(%61, %62, %63, %64, %65), scope: BiGRU/
GRU[gru] # /Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages
/torch/nn/modules/rnn.py:691:0
  %90 : Tensor[] = prim::ListConstruct(%6, %7, %8, %9, %10, %11, %12, %13, %14
, %15, %16, %17, %18, %19, %20, %21), scope: BiGRU/GRU[gru]
  %91 : bool = prim::Constant[value=1](), scope: BiGRU/GRU[gru] # /Users/vsith
am/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.p
y:682:0
  %92 : int = prim::Constant[value=2](), scope: BiGRU/GRU[gru] # /Users/vsitha
m/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.py
:682:0
  %93 : float = prim::Constant[value=0.5](), scope: BiGRU/GRU[gru] # /Users/vs
itham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rn
n.py:682:0
  %94 : bool = prim::Constant[value=0](), scope: BiGRU/GRU[gru] # /Users/vsith
am/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.p
y:682:0
  %95 : bool = prim::Constant[value=1](), scope: BiGRU/GRU[gru] # /Users/vsith
am/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/rnn.p
y:682:0
  %96 : Float(6948, 8), %hidden.1 : Float(4, 256, 4) = aten::gru(%input.3, %ba
tch_sizes, %hx, %90, %91, %92, %93, %94, %95), scope: BiGRU/GRU[gru] # /Users/
vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/modules/
rnn.py:682:0
  %99 : int = prim::Constant[value=2](), scope: BiGRU # <ipython-input-11-5f3a
e3dfaec2>:108:0
  %100 : int = prim::Constant[value=2](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:108:0
  %101 : int = prim::Constant[value=4](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:108:0
  %102 : int[] = prim::ListConstruct(%99, %100, %98, %101), scope: BiGRU
  %hidden : Float(2, 2, 256, 4) = aten::view(%hidden.1, %102), scope: BiGRU #
<ipython-input-11-5f3ae3dfaec2>:108:0
  %104 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:109:0
  %105 : int = prim::Constant[value=-1](), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:109:0
  %last_hidden.1 : Float(2, 256, 4) = aten::select(%hidden, %104, %105), scope
: BiGRU # <ipython-input-11-5f3ae3dfaec2>:109:0
  %107 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:112:0
```

```
  %108 : int[] = prim::ListConstruct(%107), scope: BiGRU
  %109 : bool = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:112:0
  %110 : int? = prim::Constant(), scope: BiGRU
  %last_hidden : Float(256, 4) = aten::sum(%last_hidden.1, %108, %109, %110),
scope: BiGRU # <ipython-input-11-5f3ae3dfaec2>:112:0
  %112 : int = prim::Constant[value=0](), scope: BiGRU # /Users/vsitham/opt/an
aconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:313:0
  %113 : int = aten::size(%batch_sizes, %112), scope: BiGRU # /Users/vsitham/o
pt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:313:
0
  %max_seq_length : Long() = prim::NumToTensor(%113), scope: BiGRU
  %115 : int = aten::Int(%max_seq_length), scope: BiGRU
  %116 : bool = prim::Constant[value=1](), scope: BiGRU # /Users/vsitham/opt/a
naconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:322:0
  %117 : float = prim::Constant[value=0](), scope: BiGRU # /Users/vsitham/opt/
anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:322:0
  %gru_out.1 : Float(256!, 28!, 8), %lengths : Long(256) = aten::_pad_packed_s
equence(%96, %batch_sizes, %116, %117, %115), scope: BiGRU # /Users/vsitham/op
t/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/utils/rnn.py:322:0
  %120 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %121 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %122 : int = prim::Constant[value=9223372036854775807](), scope: BiGRU # <ip
ython-input-11-5f3ae3dfaec2>:121:0
  %123 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %124 : Float(256!, 28!, 8) = aten::slice(%gru_out.1, %120, %121, %122, %123)
, scope: BiGRU # <ipython-input-11-5f3ae3dfaec2>:121:0
  %125 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %126 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %127 : int = prim::Constant[value=9223372036854775807](), scope: BiGRU # <ip
ython-input-11-5f3ae3dfaec2>:121:0
  %128 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %129 : Float(256!, 28!, 8) = aten::slice(%124, %125, %126, %127, %128), scop
e: BiGRU # <ipython-input-11-5f3ae3dfaec2>:121:0
  %130 : int = prim::Constant[value=2](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %131 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %132 : int = prim::Constant[value=4](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %133 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %134 : Float(256!, 28!, 4) = aten::slice(%129, %130, %131, %132, %133), scop
e: BiGRU # <ipython-input-11-5f3ae3dfaec2>:121:0
  %135 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %136 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %137 : int = prim::Constant[value=9223372036854775807](), scope: BiGRU # <ip
ython-input-11-5f3ae3dfaec2>:121:0
  %138 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
```

```
ae3dfaec2>:121:0
  %139 : Float(256!, 28!, 8) = aten::slice(%gru_out.1, %135, %136, %137, %138)
, scope: BiGRU # <ipython-input-11-5f3ae3dfaec2>:121:0
  %140 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %141 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %142 : int = prim::Constant[value=9223372036854775807](), scope: BiGRU # <ip
ython-input-11-5f3ae3dfaec2>:121:0
  %143 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %144 : Float(256!, 28!, 8) = aten::slice(%139, %140, %141, %142, %143), scop
e: BiGRU # <ipython-input-11-5f3ae3dfaec2>:121:0
  %145 : int = prim::Constant[value=2](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %146 : int = prim::Constant[value=4](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %147 : int = prim::Constant[value=9223372036854775807](), scope: BiGRU # <ip
ython-input-11-5f3ae3dfaec2>:121:0
  %148 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %149 : Float(256!, 28!, 4) = aten::slice(%144, %145, %146, %147, %148), scop
e: BiGRU # <ipython-input-11-5f3ae3dfaec2>:121:0
  %150 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:121:0
  %gru_out : Float(256!, 28!, 4) = aten::add(%134, %149, %150), scope: BiGRU #
<ipython-input-11-5f3ae3dfaec2>:121:0
  %152 : int = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:126:0
  %153 : int = prim::Constant[value=2](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:126:0
  %154 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:126:0
  %155 : int[] = prim::ListConstruct(%152, %153, %154), scope: BiGRU
  %input.4 : Float(256, 4!, 28!) = aten::permute(%gru_out, %155), scope: BiGRU
# <ipython-input-11-5f3ae3dfaec2>:126:0
  %157 : int = prim::Constant[value=1](), scope: BiGRU # /Users/vsitham/opt/an
aconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/functional.py:664:0
  %158 : int[] = prim::ListConstruct(%157), scope: BiGRU
  %159 : Float(256, 4, 1), %160 : Long(256, 4, 1) = aten::adaptive_max_pool1d(
%input.4, %158), scope: BiGRU # /Users/vsitham/opt/anaconda3/envs/py3.7/lib/py
thon3.7/site-packages/torch/nn/functional.py:664:0
  %162 : int = prim::Constant[value=-1](), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:126:0
  %163 : int[] = prim::ListConstruct(%161, %162), scope: BiGRU
  %max_pool : Float(256, 4) = aten::view(%159, %163), scope: BiGRU # <ipython-
input-11-5f3ae3dfaec2>:126:0
  %165 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:131:0
  %166 : int[] = prim::ListConstruct(%165), scope: BiGRU
  %167 : bool = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:131:0
  %168 : int? = prim::Constant(), scope: BiGRU
  %169 : Float(256, 4) = aten::sum(%gru_out, %166, %167, %168), scope: BiGRU #
<ipython-input-11-5f3ae3dfaec2>:131:0
  %170 : int = prim::Constant[value=-1](), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:131:0
```

```
  %171 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:131:0
  %172 : int[] = prim::ListConstruct(%170, %171), scope: BiGRU
  %173 : Long(256, 1) = aten::view(%lengths, %172), scope: BiGRU # <ipython-in
put-11-5f3ae3dfaec2>:131:0
  %174 : Device = prim::Constant[value="cpu"](), scope: BiGRU # <ipython-input
-11-5f3ae3dfaec2>:131:0
  %175 : int = prim::Constant[value=6](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:131:0
  %176 : bool = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:131:0
  %177 : bool = prim::Constant[value=0](), scope: BiGRU # <ipython-input-11-5f
3ae3dfaec2>:131:0
  %178 : Float(256, 1) = aten::to(%173, %174, %175, %176, %177), scope: BiGRU
# <ipython-input-11-5f3ae3dfaec2>:131:0
  %avg_pool : Float(256, 4) = aten::div(%169, %178), scope: BiGRU # <ipython-i
nput-11-5f3ae3dfaec2>:131:0
  %180 : Tensor[] = prim::ListConstruct(%last_hidden, %max_pool, %avg_pool), s
cope: BiGRU
  %181 : int = prim::Constant[value=1](), scope: BiGRU # <ipython-input-11-5f3
ae3dfaec2>:134:0
  %input.5 : Float(256, 12) = aten::cat(%180, %181), scope: BiGRU # <ipython-i
nput-11-5f3ae3dfaec2>:134:0
  %183 : Float(12!, 2!) = aten::t(%weight), scope: BiGRU/Linear[linear] # /Use
rs/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/funct
ional.py:1369:0
  %184 : int = prim::Constant[value=1](), scope: BiGRU/Linear[linear] # /Users
/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/functio
nal.py:1369:0
  %185 : int = prim::Constant[value=1](), scope: BiGRU/Linear[linear] # /Users
/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/functio
nal.py:1369:0
  %input : Float(256, 2) = aten::addmm(%bias, %input.5, %183, %184, %185), sco
pe: BiGRU/Linear[linear] # /Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3
.7/site-packages/torch/nn/functional.py:1369:0
  %187 : int = prim::Constant[value=-1](), scope: BiGRU # /Users/vsitham/opt/a
naconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/functional.py:1316:0
  %188 : int? = prim::Constant(), scope: BiGRU
  %189 : Float(256, 2) = aten::log_softmax(%input, %187, %188), scope: BiGRU #
/Users/vsitham/opt/anaconda3/envs/py3.7/lib/python3.7/site-packages/torch/nn/f
unctional.py:1316:0
  return (%189)
```

The generalization error

In [27]:

```
# Import the dataset. Use clean_review and label columns
test_dataset = pd.read_csv('dataset/drugreview_feat_clean/test_feat_clean.csv
                           usecols=['clean_review', 'rating'])

# Change columns order
test_dataset['label'] = test_dataset.rating >= 5
test_dataset = test_dataset[['clean_review', 'label']]
```

In [28]:
```
test_dataset = test_dataset.dropna()
test_dataset.head()
```

Out[28]:

	clean_review	label
2	given sample doctor mg hours lower abdominal g...	False
3	given medication post hysteroscopy suffered se...	True
4	loperamide helpful diarrhea fewer caplets help...	True
10	use claritin d seasonal allergies started taki...	True
15	worked immediate effects noticeable long term	True

In [29]:
```
test_iterator = BatchIterator(test_dataset, batch_size=256, vocab_created=Fal
                              word2index=train_iterator.word2index, sos_token
                              unk_token='<UNK>', pad_token='<PAD>', min_word_
                              max_seq_len=0.9, use_pretrained_vectors=False,
                              glove_name='glove.6B.100d.txt', weights_file_na
```

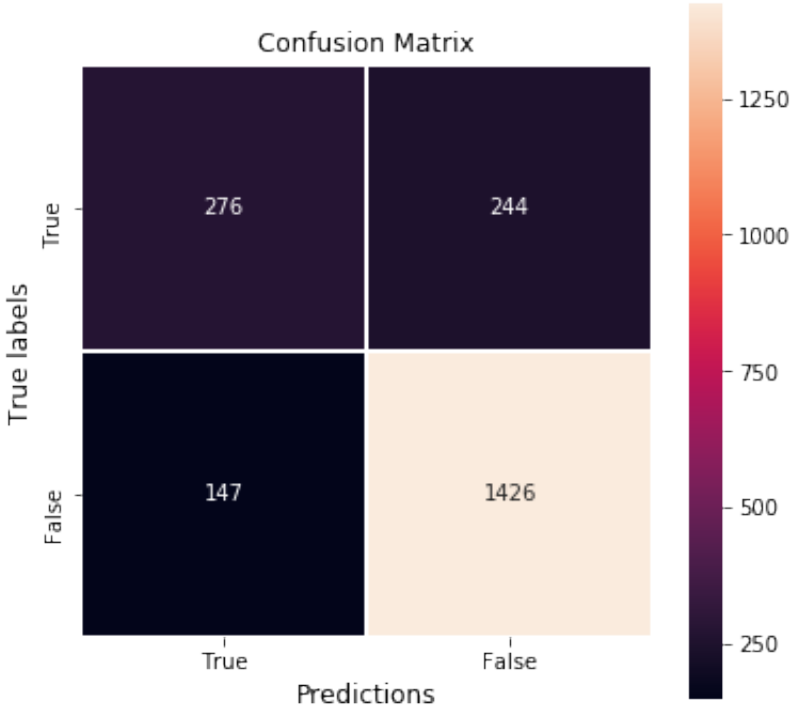
```
Trimmed vocabulary using as minimum count threashold: count = 3.00
3069/8377 tokens has been retained
Trimmed input strings vocabulary
Trimmed input sequences lengths to the length of: 54
Mapped words to indices
Batches created
```

In [30]:
```
_, test_avg_loss, test_accuracy, test_conf_matrix = model.evaluate_model(test
```

In [31]:
```
print('Test accuracy: {:.3f}. Test error: {:.3f}'.format(test_accuracy, test_
```

```
Test accuracy: 0.813. Test error: 0.429
```

In [32]:
```
# Confusion matrix
plt.figure(figsize=(6,6))
ax = sns.heatmap(test_conf_matrix, fmt='d', annot=True, linewidths=1, square='
ax.set_xlabel('Predictions', size=12)
ax.set_ylabel('True labels', size=12)
ax.set_title('Confusion Matrix', size=12);
ax.xaxis.set_ticklabels(['True', 'False'])
ax.yaxis.set_ticklabels(['True', 'False'])
ax.set_ylim(2,0)
plt.show()
```



The generalization accuracy of the biGRU model equals 0.813. As we can see on the above plot of the confusion matrix the both, positive and negative classes were similarly numerous, and the prediction mistakes amount (TN, FP) is also very similar, so model learned both classes in the same detail.

In [ ]: