In [1]:

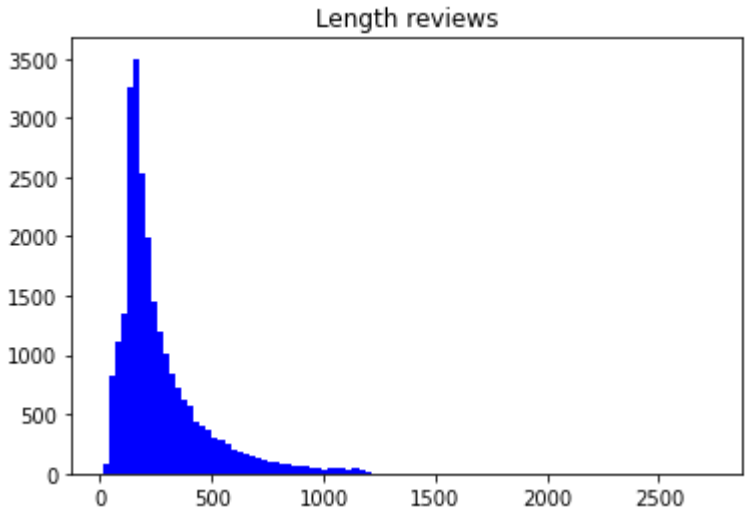
```
import tensorflow as tf
import pandas as pd
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
tf.random.set_seed(1234)
```

In [2]:

```
import sys
sys.path.insert(1,'/data/')
from data_utils import parse_imdb_sequence
```

In [3]:

```
length_reviews = pickle.load(open('./data/length_reviews.pkl', 'rb'))
pd.DataFrame(length_reviews, columns=['Length reviews']).hist(bins=100, color='blu
e');
plt.grid(False);
```



In [4]:

```
full_dataset  = tf.data.TFRecordDataset('./data/train.tfrecords')
full_dataset  = full_dataset.shuffle(buffer_size=10000)

DATASET_SIZE  = sum(1 for _ in full_dataset)
print(DATASET_SIZE)

train_size = int(0.85 * DATASET_SIZE)
val_size   = int(0.15 * DATASET_SIZE)

train_dataset = full_dataset.take(train_size)
val_dataset   = full_dataset.take(val_size)

train_dataset_size  = sum(1 for _ in train_dataset)
print(train_dataset_size)

val_dataset_size  = sum(1 for _ in val_dataset)
print(val_dataset_size)

train_dataset = train_dataset.map(parse_imdb_sequence).shuffle(buffer_size=10000)
train_dataset = train_dataset.padded_batch(512, padded_shapes=([None],[],[]))

val_dataset = val_dataset.map(parse_imdb_sequence).shuffle(buffer_size=10000)
val_dataset = val_dataset.padded_batch(128, padded_shapes=([None],[],[]))

test_dataset = tf.data.TFRecordDataset('./data/test.tfrecords')
test_dataset = test_dataset.map(parse_imdb_sequence).shuffle(buffer_size=10000)
test_dataset = test_dataset.padded_batch(512, padded_shapes=([None],[],[]))
```

```
2022-03-28 23:07:48.281394: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.290323: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.290656: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.292315: I tensorflow/core/platform/cpu_feature_gua
rd.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
2022-03-28 23:07:48.293619: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.293960: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.294208: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.617958: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.618318: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.618579: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-28 23:07:48.618829: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1525] Created device /job:localhost/replica:0/task:0/device:G
PU:0 with 6980 MB memory:  -> device: 0, name: NVIDIA GeForce GTX 107
0, pci bus id: 0000:01:00.0, compute capability: 6.1


25000
21250
3750
```

In [5]:

```
# Read the word vocabulary
word2idx = pickle.load(open('./data/word2idx.pkl', 'rb'))
```

In [6]:

```
class RNNModel(tf.keras.Model):
    def __init__(self, embedding_size=100, cell_size=64, dense_size=128,
                 num_classes=2, vocabulary_size=None, rnn_cell='lstm',
                 device='cpu:0', checkpoint_directory=None):
        ''' Define the parameterized layers used during forward-pass, the device
            where you would like to run the computation on and the checkpoint
            directory. Additionaly, you can also modify the default size of the
            network.

            Args:
                embedding_size: the size of the word embedding.
                cell_size: RNN cell size.
                dense_size: the size of the dense layer.
                num_classes: the number of labels in the network.
                vocabulary_size: the size of the word vocabulary.
                rnn_cell: string, either 'lstm' or 'ugrnn'.
                device: string, 'cpu:n' or 'gpu:n' (n can vary). Default, 'cpu:0'.
                checkpoint_directory: the directory where you would like to save o
r
                                      restore a model.
        '''
        super(RNNModel, self).__init__()

        # Weights initializer function
        w_initializer = tf.compat.v1.keras.initializers.glorot_uniform()

        # Biases initializer function
        b_initializer = tf.zeros_initializer()

        # Initialize weights for word embeddings
        self.embeddings = tf.keras.layers.Embedding(vocabulary_size, embedding_siz
e,
                                                    embeddings_initializer=w_initi
alizer)

        # Dense layer initialization
        self.dense_layer = tf.keras.layers.Dense(dense_size, activation=tf.nn.relu
,
                                                 kernel_initializer=w_initializer,
                                                 bias_initializer=b_initializer)

        # Predictions layer initialization
        self.pred_layer = tf.keras.layers.Dense(num_classes, activation=None,
                                                kernel_initializer=w_initializer,
                                                bias_initializer=b_initializer)

        # Basic LSTM cell
        if rnn_cell=='lstm':
            self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicLSTMCell(cell_size)
        # Else RNN cell
        else:
            self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicRNNCell(cell_size)
```

```
        # Define the device
        self.device = device

        # Define the checkpoint directory
        self.checkpoint_directory = checkpoint_directory

    def predict(self, X, seq_length, is_training):
        '''
        Predicts the probability of each class, based on the input sample.

        Args:
            X: 2D tensor of shape (batch_size, time_steps).
            seq_length: the length of each sequence in the batch.
            is_training: Boolean. Either the network is predicting in
                         training mode or not.
        '''

        # Get the number of samples within a batch
        num_samples = tf.shape(X)[0]

        # Initialize LSTM cell state with zeros
        state = self.rnn_cell.zero_state(num_samples, dtype=tf.float32)

        # Get the embedding of each word in the sequence
        embedded_words = self.embeddings(X)

        # Unstack the embeddings
        unstacked_embeddings = tf.unstack(embedded_words, axis=1)

        # Iterate through each timestep and append the predictions
        outputs = []
        for input_step in unstacked_embeddings:
            output, state = self.rnn_cell(input_step, state)
            outputs.append(output)

        # Stack outputs to (batch_size, time_steps, cell_size)
        outputs = tf.stack(outputs, axis=1)

        # Extract the output of the last time step, of each sample
        idxs_last_output = tf.stack([tf.range(num_samples),
                                     tf.cast(seq_length-1, tf.int32)], axis=1)
        final_output = tf.gather_nd(outputs, idxs_last_output)

        # Add dropout for regularization
        #dropped_output = tf.compat.v1.layers.Dropout(final_output, rate=0.3, trai
ning=is_training)

        # Pass the last cell state through a dense layer (ReLU activation)
        dense = self.dense_layer(final_output)

        # Compute the unnormalized log probabilities
        logits = self.pred_layer(dense)
        return logits

    def loss_fn(self, X, y, seq_length, is_training):
        """ Defines the loss function used during
```

```
            training.
        """
        preds = self.predict(X, seq_length, is_training)
        loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=pre
ds)
        return loss

    def grads_fn(self, X, y, seq_length, is_training):
        """ Dynamically computes the gradients of the loss value
            with respect to the parameters of the model, in each
            forward pass.
        """
        with tf.GradientTape() as tape:
            loss = self.loss_fn(X, y, seq_length, is_training)
        return tape.gradient(loss, self.variables)

    def restore_model(self):
        """ Function to restore trained model.
        """
        with tf.device(self.device):
            # Run the model once to initialize variables
            dummy_input = tf.constant(tf.zeros((1,1)))
            dummy_length = tf.constant(1, shape=(1,))
            dummy_pred = self.predict(dummy_input, dummy_length, False)
            # Restore the variables of the model
            saver = tf.compat.v1.train.Saver(self.variables)
            saver.restore(tf.train.latest_checkpoint
                          (self.checkpoint_directory))

    def save_model(self, global_step=0):
        """ Function to save trained model.
        """
        tf.compat.v1.train.Saver(self.variables).save(save_path=self.checkpoint_di
rectory,
                                      global_step=global_step)

    def fit(self, training_data, eval_data, test_data, optimizer, num_epochs=500,
            early_stopping_rounds=10, verbose=10, train_from_scratch=False):
        """ Function to train the model, using the selected optimizer and
            for the desired number of epochs. You can either train from scratch
            or load the latest model trained. Early stopping is used in order to
            mitigate the risk of overfitting the network.

            Args:
                training_data: the data you would like to train the model on.
                                Must be in the tf.data.Dataset format.
                eval_data: the data you would like to evaluate the model on.
                            Must be in the tf.data.Dataset format.
                optimizer: the optimizer used during training.
                num_epochs: the maximum number of iterations you would like to
                            train the model.
                early_stopping_rounds: stop training if the accuracy on the eval
                                       dataset does not increase after n epochs.
                verbose: int. Specify how often to print the loss value of the net
work.
                train_from_scratch: boolean. Whether to initialize variables of th
```

```
e
                                    the last trained model or initialize them
                                    randomly.
        """

        if train_from_scratch==False:
            self.restore_model()

        # Initialize best_acc. This variable will store the highest accuracy
        # on the eval dataset.
        best_acc = 0

        # Initialize classes to update the mean accuracy of train and eval
        train_acc = tf.keras.metrics.Accuracy('train_acc')
        eval_acc = tf.keras.metrics.Accuracy('eval_acc')
        test_acc = tf.keras.metrics.Accuracy('test_acc')
        # Initialize dictionary to store the accuracy history
        self.history = {}
        self.history['train_acc'] = []
        self.history['eval_acc'] = []
        self.history['test_acc'] = []

        # Begin training
        with tf.device(self.device):
            for i in range(num_epochs):
                # Training with gradient descent
                for step, (X, y, seq_length) in enumerate(training_data):
                    grads = self.grads_fn(X, y, seq_length, True)
                    optimizer.apply_gradients(zip(grads, self.variables))

                # Check accuracy train dataset
                for step, (X, y, seq_length) in enumerate(training_data):
                    logits = self.predict(X, seq_length, False)
                    preds = tf.argmax(logits, axis=1)
                    train_acc(preds, y)
                self.history['train_acc'].append(train_acc.result().numpy())
                # Reset metrics
                train_acc.reset_states()

                # Check accuracy eval dataset
                for step, (X, y, seq_length) in enumerate(eval_data):
                    logits = self.predict(X, seq_length, False)
                    preds = tf.argmax(logits, axis=1)
                    eval_acc(preds, y)
                self.history['eval_acc'].append(eval_acc.result().numpy())
                # Reset metrics
                eval_acc.reset_states()

                # Print train and eval accuracy
                if (i==0) | ((i+1)%verbose==0):
                    print('Train accuracy at epoch %d: ' %(i+1), self.history['tra
in_acc'][-1])
                    print('Eval accuracy at epoch %d: ' %(i+1), self.history['eval
_acc'][-1])

                # Check for early stopping
```

```
            if self.history['eval_acc'][-1]>best_acc:
                best_acc = self.history['eval_acc'][-1]
                count = early_stopping_rounds
            else:
                count -= 1
            if count==0:
                break

        for step, (X, y, seq_length) in enumerate(test_data):
            logits = self.predict(X, seq_length, False)
            preds = tf.argmax(logits, axis=1)
            test_acc(preds, y)
        self.history['test_acc'].append(test_acc.result().numpy())
        print('Test accuracy: ', self.history['test_acc'][-1])
```

In [7]:

```
# Specify the path where you want to save/restore the trained variables.
checkpoint_directory = './models_checkpoints/ImdbRNN1/'

# Use the GPU if available.
device = 'gpu:0'

# Define optimizer.
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-4)

# Instantiate model. This doesn't initialize the variables yet.
lstm_model1 = RNNModel(vocabulary_size=len(word2idx), device=device,
                       checkpoint_directory=checkpoint_directory)

# Train model
lstm_model1.fit(train_dataset, val_dataset, test_dataset, optimizer, num_epochs=20
,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(lstm_model1)
save_path = checkpoint.save(checkpoint_directory)

################################################################################
############################

# Define optimizer.
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-4)

# Instantiate model. This doesn't initialize the variables yet.
ugrnn_model1 = RNNModel(vocabulary_size=len(word2idx), rnn_cell='ugrnn',
                        device=device, checkpoint_directory=checkpoint_directory)

# Train model
ugrnn_model1.fit(train_dataset,  val_dataset, test_dataset, optimizer, num_epochs=
20,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(ugrnn_model1)
save_path = checkpoint.save(checkpoint_directory)

################################################################################
############################

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
ax1.plot(range(len(lstm_model1.history['train_acc'])), lstm_model1.history['train_
acc'],
         label='LSTM Train Accuracy');
ax1.plot(range(len(lstm_model1.history['eval_acc'])), lstm_model1.history['eval_ac
c'],
         label='LSTM Test Accuracy');
ax2.plot(range(len(ugrnn_model1.history['train_acc'])), ugrnn_model1.history['trai
n_acc'],
```

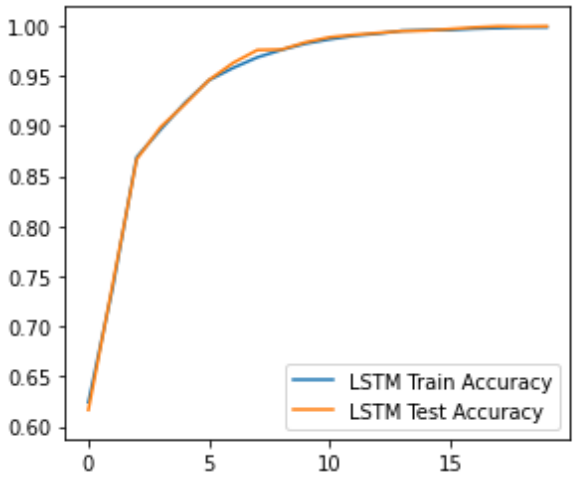
```
        label='UGRNN Train Accuracy');
ax2.plot(range(len(ugrnn_model1.history['eval_acc'])), ugrnn_model1.history['eval_
acc'],
        label='UGRNN Test Accuracy');
ax1.legend();
ax2.legend();
```

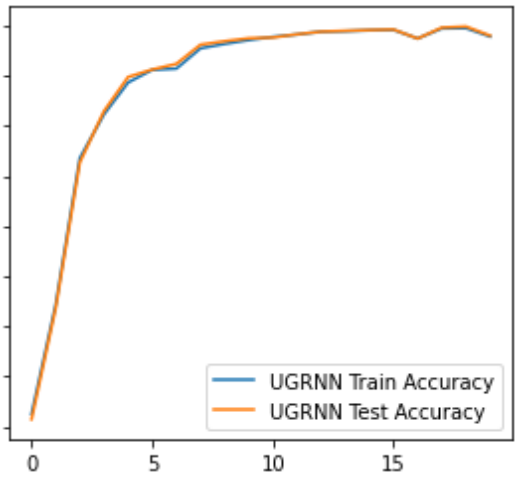
```
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicLSTMCel
l object at 0x7fec2908f850>: Note that this cell is not optimized for 
performance. Please use tf.contrib.cudnn_rnn.CudnnLSTM for better perf
ormance on GPU.

/tmp/ipykernel_7766/2071740490.py:45: UserWarning: `tf.nn.rnn_cell.Bas
icLSTMCell` is deprecated and will be removed in a future version. Thi
s class is equivalent as `tf.keras.layers.LSTMCell`, and will be repla
ced by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicLSTMCell(cell_size)
/home/skanda/Softwares/miniconda3/lib/python3.9/site-packages/keras/la
yers/legacy_rnn/rnn_cell_impl.py:754: UserWarning: `layer.add_variable
` is deprecated and will be removed in a future version. Please use `l
ayer.add_weight` method instead.
  self._kernel = self.add_variable(
/home/skanda/Softwares/miniconda3/lib/python3.9/site-packages/keras/la
yers/legacy_rnn/rnn_cell_impl.py:757: UserWarning: `layer.add_variable
` is deprecated and will be removed in a future version. Please use `l
ayer.add_weight` method instead.
  self._bias = self.add_variable(
```

```
Train accuracy at epoch 1:  0.6242353
Eval accuracy at epoch 1:  0.61653334
Train accuracy at epoch 2:  0.73778826
Eval accuracy at epoch 2:  0.7410667
Train accuracy at epoch 3:  0.8689412
Eval accuracy at epoch 3:  0.86693335
Train accuracy at epoch 4:  0.89675295
Eval accuracy at epoch 4:  0.8992
Train accuracy at epoch 5:  0.9232941
Eval accuracy at epoch 5:  0.92186666
Train accuracy at epoch 6:  0.9458824
Eval accuracy at epoch 6:  0.9461333
Train accuracy at epoch 7:  0.9581647
Eval accuracy at epoch 7:  0.9632
Train accuracy at epoch 8:  0.9687059
Eval accuracy at epoch 8:  0.976
Train accuracy at epoch 9:  0.9760941
Eval accuracy at epoch 9:  0.97653335
Train accuracy at epoch 10:  0.9824
Eval accuracy at epoch 10:  0.98373336
Train accuracy at epoch 11:  0.9867294
Eval accuracy at epoch 11:  0.9888
Train accuracy at epoch 12:  0.9902118
Eval accuracy at epoch 12:  0.9912
Train accuracy at epoch 13:  0.9923294
Eval accuracy at epoch 13:  0.99306667
Train accuracy at epoch 14:  0.99524707
Eval accuracy at epoch 14:  0.9949333
Train accuracy at epoch 15:  0.99595296
Eval accuracy at epoch 15:  0.99546665
Train accuracy at epoch 16:  0.9960471
Eval accuracy at epoch 16:  0.9970667
Train accuracy at epoch 17:  0.9972706
Eval accuracy at epoch 17:  0.99866664
Train accuracy at epoch 18:  0.9979294
Eval accuracy at epoch 18:  0.9997333
Train accuracy at epoch 19:  0.9986824
Eval accuracy at epoch 19:  0.99946666
Train accuracy at epoch 20:  0.9987765
Eval accuracy at epoch 20:  0.9997333
Test accuracy:  0.8564
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicRNNCell
object at 0x7febc0368520>: Note that this cell is not optimized for pe
rformance. Please use tf.contrib.cudnn_rnn.CudnnRNNTanh for better per
formance on GPU.
```

```
/tmp/ipykernel_7766/2071740490.py:48: UserWarning: `tf.nn.rnn_cell.Bas
icRNNCell` is deprecated and will be removed in a future version. This
class is equivalent as `tf.keras.layers.SimpleRNNCell`, and will be re
placed by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicRNNCell(cell_size)
/home/skanda/Softwares/miniconda3/lib/python3.9/site-packages/keras/la
yers/legacy_rnn/rnn_cell_impl.py:457: UserWarning: `layer.add_variable
` is deprecated and will be removed in a future version. Please use `l
ayer.add_weight` method instead.
  self._kernel = self.add_variable(
/home/skanda/Softwares/miniconda3/lib/python3.9/site-packages/keras/la
yers/legacy_rnn/rnn_cell_impl.py:460: UserWarning: `layer.add_variable
` is deprecated and will be removed in a future version. Please use `l
ayer.add_weight` method instead.
  self._bias = self.add_variable(
```

```
Train accuracy at epoch 1:  0.61242354
Eval accuracy at epoch 1:  0.6072
Train accuracy at epoch 2:  0.72145885
Eval accuracy at epoch 2:  0.7184
Train accuracy at epoch 3:  0.86804706
Eval accuracy at epoch 3:  0.86373335
Train accuracy at epoch 4:  0.9117647
Eval accuracy at epoch 4:  0.9149333
Train accuracy at epoch 5:  0.94357646
Eval accuracy at epoch 5:  0.94906664
Train accuracy at epoch 6:  0.9562353
Eval accuracy at epoch 6:  0.9565333
Train accuracy at epoch 7:  0.95741177
Eval accuracy at epoch 7:  0.96213335
Train accuracy at epoch 8:  0.9776941
Eval accuracy at epoch 8:  0.9813333
Train accuracy at epoch 9:  0.98211765
Eval accuracy at epoch 9:  0.9848
Train accuracy at epoch 10:  0.98597646
Eval accuracy at epoch 10:  0.9877333
Train accuracy at epoch 11:  0.9891294
Eval accuracy at epoch 11:  0.9885333
Train accuracy at epoch 12:  0.9915294
Eval accuracy at epoch 12:  0.9917333
Train accuracy at epoch 13:  0.99435294
Eval accuracy at epoch 13:  0.9944
Train accuracy at epoch 14:  0.9948235
Eval accuracy at epoch 14:  0.9952
Train accuracy at epoch 15:  0.99576473
Eval accuracy at epoch 15:  0.996
Train accuracy at epoch 16:  0.9961412
Eval accuracy at epoch 16:  0.99653333
Train accuracy at epoch 17:  0.9875294
Eval accuracy at epoch 17:  0.9874667
Train accuracy at epoch 18:  0.99755293
Eval accuracy at epoch 18:  0.9984
Train accuracy at epoch 19:  0.9978353
Eval accuracy at epoch 19:  0.99946666
Train accuracy at epoch 20:  0.9895059
Eval accuracy at epoch 20:  0.9904
Test accuracy:  0.84372
```


1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
LSTM Train Accuracy
LSTM Test Accuracy


0
5
10
15
UGRNN Train Accuracy
UGRNN Test Accuracy

In [8]:

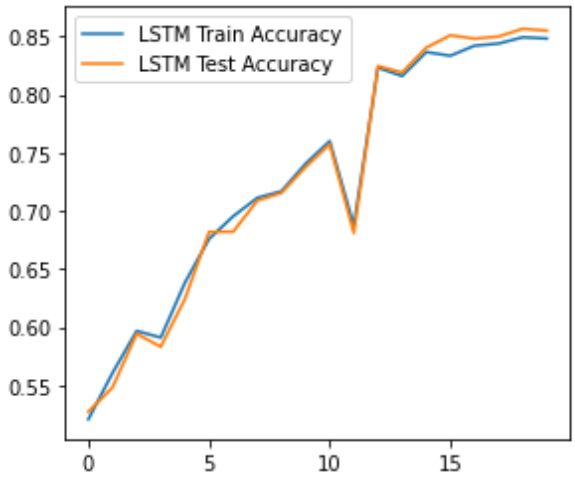
```
# Specify the path where you want to save/restore the trained variables.
checkpoint_directory = 'models_checkpoints/ImdbRNN2/'

# Use the GPU if available.
device = 'gpu:0'

# Define optimizer.
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-5)

# Instantiate model. This doesn't initialize the variables yet.
lstm_model2 = RNNModel(vocabulary_size=len(word2idx), device=device,
                       checkpoint_directory=checkpoint_directory)

# Train model
lstm_model2.fit(train_dataset, val_dataset, test_dataset, optimizer, num_epochs=20
,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(lstm_model2)
save_path = checkpoint.save(checkpoint_directory)


################################################################################
############################

# Define optimizer.
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-5)

# Instantiate model. This doesn't initialize the variables yet.
ugrnn_model2 = RNNModel(vocabulary_size=len(word2idx), rnn_cell='ugrnn',
                        device=device, checkpoint_directory=checkpoint_directory)

# Train model
ugrnn_model2.fit(train_dataset,  val_dataset, test_dataset, optimizer, num_epochs=
20,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(ugrnn_model2)
save_path = checkpoint.save(checkpoint_directory)

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
ax1.plot(range(len(lstm_model2.history['train_acc'])), lstm_model2.history['train_
acc'],
         label='LSTM Train Accuracy');
ax1.plot(range(len(lstm_model2.history['eval_acc'])), lstm_model2.history['eval_ac
c'],
         label='LSTM Test Accuracy');
ax2.plot(range(len(ugrnn_model2.history['train_acc'])), ugrnn_model2.history['trai
n_acc'],
         label='UGRNN Train Accuracy');
ax2.plot(range(len(ugrnn_model2.history['eval_acc'])), ugrnn_model2.history['eval_
```

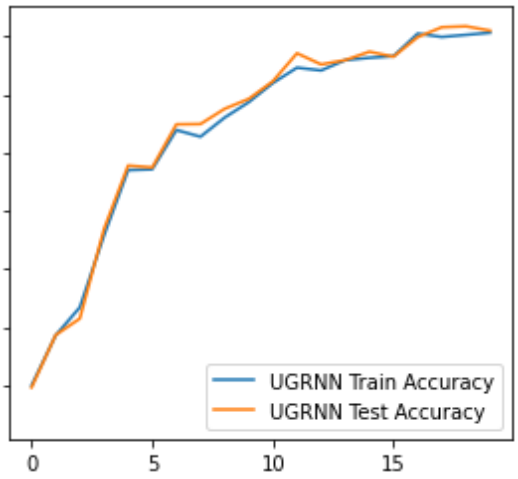
```
acc'],
         label='UGRNN Test Accuracy');
ax1.legend();
ax2.legend();
```

```
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicLSTMCel
l object at 0x7fec291658e0>: Note that this cell is not optimized for 
performance. Please use tf.contrib.cudnn_rnn.CudnnLSTM for better perf
ormance on GPU.

/tmp/ipykernel_7766/2071740490.py:45: UserWarning: `tf.nn.rnn_cell.Bas
icLSTMCell` is deprecated and will be removed in a future version. Thi
s class is equivalent as `tf.keras.layers.LSTMCell`, and will be repla
ced by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicLSTMCell(cell_size)
```

```
Train accuracy at epoch 1:  0.5209412
Eval accuracy at epoch 1:  0.52746665
Train accuracy at epoch 2:  0.5607529
Eval accuracy at epoch 2:  0.54826665
Train accuracy at epoch 3:  0.5968
Eval accuracy at epoch 3:  0.5944
Train accuracy at epoch 4:  0.59115297
Eval accuracy at epoch 4:  0.5832
Train accuracy at epoch 5:  0.6383059
Eval accuracy at epoch 5:  0.6242667
Train accuracy at epoch 6:  0.6759059
Eval accuracy at epoch 6:  0.68186665
Train accuracy at epoch 7:  0.69534117
Eval accuracy at epoch 7:  0.6821333
Train accuracy at epoch 8:  0.7112
Eval accuracy at epoch 8:  0.7088
Train accuracy at epoch 9:  0.7168941
Eval accuracy at epoch 9:  0.7154667
Train accuracy at epoch 10:  0.7405647
Eval accuracy at epoch 10:  0.7376
Train accuracy at epoch 11:  0.76
Eval accuracy at epoch 11:  0.75733334
Train accuracy at epoch 12:  0.6872941
Eval accuracy at epoch 12:  0.6810667
Train accuracy at epoch 13:  0.82305884
Eval accuracy at epoch 13:  0.82453334
Train accuracy at epoch 14:  0.81590587
Eval accuracy at epoch 14:  0.8189333
Train accuracy at epoch 15:  0.83651763
Eval accuracy at epoch 15:  0.84026664
Train accuracy at epoch 16:  0.83331764
Eval accuracy at epoch 16:  0.8509333
Train accuracy at epoch 17:  0.84197646
Eval accuracy at epoch 17:  0.848
Train accuracy at epoch 18:  0.84381175
Eval accuracy at epoch 18:  0.8498667
Train accuracy at epoch 19:  0.84922355
Eval accuracy at epoch 19:  0.85653335
Train accuracy at epoch 20:  0.8481412
Eval accuracy at epoch 20:  0.8549333
Test accuracy:  0.7908
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicRNNCell
object at 0x7febc03732b0>: Note that this cell is not optimized for pe
rformance. Please use tf.contrib.cudnn_rnn.CudnnRNNTanh for better per
formance on GPU.

/tmp/ipykernel_7766/2071740490.py:48: UserWarning: `tf.nn.rnn_cell.Bas
icRNNCell` is deprecated and will be removed in a future version. This
class is equivalent as `tf.keras.layers.SimpleRNNCell`, and will be re
placed by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicRNNCell(cell_size)
```

```
Train accuracy at epoch 1:  0.54992944
Eval accuracy at epoch 1:  0.54826665
Train accuracy at epoch 2:  0.5930353
Eval accuracy at epoch 2:  0.5933333
Train accuracy at epoch 3:  0.6171294
Eval accuracy at epoch 3:  0.60746664
Train accuracy at epoch 4:  0.67877644
Eval accuracy at epoch 4:  0.6842667
Train accuracy at epoch 5:  0.73505884
Eval accuracy at epoch 5:  0.7389333
Train accuracy at epoch 6:  0.7357647
Eval accuracy at epoch 6:  0.73733336
Train accuracy at epoch 7:  0.76950586
Eval accuracy at epoch 7:  0.7744
Train accuracy at epoch 8:  0.7637647
Eval accuracy at epoch 8:  0.77466667
Train accuracy at epoch 9:  0.78018826
Eval accuracy at epoch 9:  0.7877333
Train accuracy at epoch 10:  0.7934588
Eval accuracy at epoch 10:  0.7962667
Train accuracy at epoch 11:  0.80988234
Eval accuracy at epoch 11:  0.8114667
Train accuracy at epoch 12:  0.8231059
Eval accuracy at epoch 12:  0.8354667
Train accuracy at epoch 13:  0.8207059
Eval accuracy at epoch 13:  0.82586664
Train accuracy at epoch 14:  0.8296
Eval accuracy at epoch 14:  0.8296
Train accuracy at epoch 15:  0.83157647
Eval accuracy at epoch 15:  0.8368
Train accuracy at epoch 16:  0.8329412
Eval accuracy at epoch 16:  0.83253336
Train accuracy at epoch 17:  0.85237646
Eval accuracy at epoch 17:  0.84933335
Train accuracy at epoch 18:  0.8495059
Eval accuracy at epoch 18:  0.85786664
Train accuracy at epoch 19:  0.8512941
Eval accuracy at epoch 19:  0.85866666
Train accuracy at epoch 20:  0.8531294
Eval accuracy at epoch 20:  0.8549333
Test accuracy:  0.75392
```


LSTM Train Accuracy
LSTM Test Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
5
10
15


UGRNN Train Accuracy
UGRNN Test Accuracy
0
5
10
15

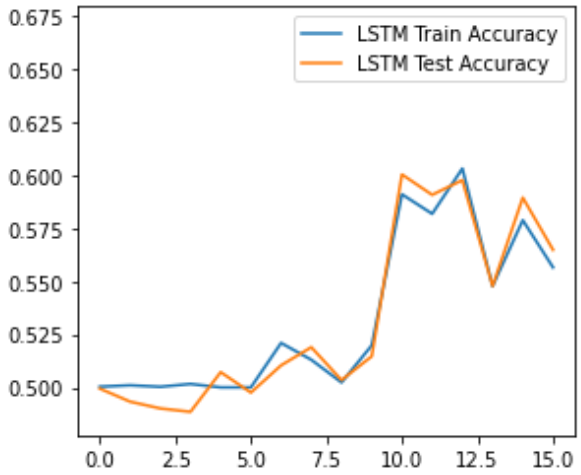
In [9]:

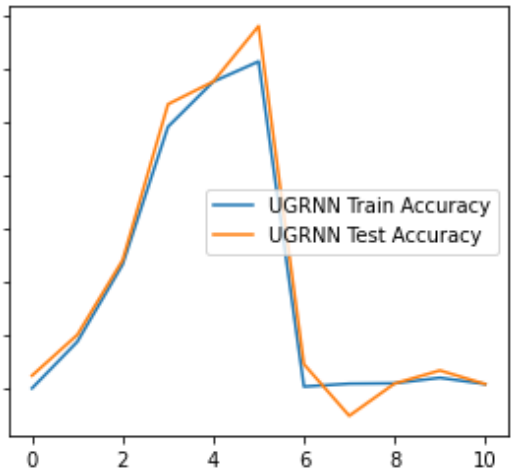
```
# Specify the path where you want to save/restore the trained variables.
checkpoint_directory = './models_checkpoints/ImdbRNN3/'

# Use the GPU if available.
device = 'gpu:0'

# Define optimizer.
optimizer = tf.compat.v1.train.GradientDescentOptimizer(learning_rate=1e-3)

# Instantiate model. This doesn't initialize the variables yet.
lstm_model3 = RNNModel(vocabulary_size=len(word2idx), device=device,
                       checkpoint_directory=checkpoint_directory)

# Train model
lstm_model3.fit(train_dataset, val_dataset, test_dataset, optimizer, num_epochs=20
,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(lstm_model3)
save_path = checkpoint.save(checkpoint_directory)

################################################################################
############################

# Define optimizer.
optimizer = tf.compat.v1.train.GradientDescentOptimizer(learning_rate=1e-3)

# Instantiate model. This doesn't initialize the variables yet.
ugrnn_model3 = RNNModel(vocabulary_size=len(word2idx), rnn_cell='ugrnn',
                        device=device, checkpoint_directory=checkpoint_directory)

# Train model
ugrnn_model3.fit(train_dataset,  val_dataset, test_dataset, optimizer, num_epochs=
20,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(ugrnn_model3)
save_path = checkpoint.save(checkpoint_directory)

################################################################################
############################

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
ax1.plot(range(len(lstm_model3.history['train_acc'])), lstm_model3.history['train_
acc'],
         label='LSTM Train Accuracy');
ax1.plot(range(len(lstm_model3.history['eval_acc'])), lstm_model3.history['eval_ac
c'],
         label='LSTM Test Accuracy');
ax2.plot(range(len(ugrnn_model3.history['train_acc'])), ugrnn_model3.history['trai
n_acc'],
```

```
        label='UGRNN Train Accuracy');
ax2.plot(range(len(ugrnn_model3.history['eval_acc'])), ugrnn_model3.history['eval_
acc'],
        label='UGRNN Test Accuracy');
ax1.legend();
ax2.legend();
```

```
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicLSTMCel
l object at 0x7febc0377bb0>: Note that this cell is not optimized for 
performance. Please use tf.contrib.cudnn_rnn.CudnnLSTM for better perf
ormance on GPU.

/tmp/ipykernel_7766/2071740490.py:45: UserWarning: `tf.nn.rnn_cell.Bas
icLSTMCell` is deprecated and will be removed in a future version. Thi
s class is equivalent as `tf.keras.layers.LSTMCell`, and will be repla
ced by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicLSTMCell(cell_size)

Train accuracy at epoch 1:  0.50037646
Eval accuracy at epoch 1:  0.49946666
Train accuracy at epoch 2:  0.5010353
Eval accuracy at epoch 2:  0.49333334
Train accuracy at epoch 3:  0.50032943
Eval accuracy at epoch 3:  0.49013335
Train accuracy at epoch 4:  0.50164706
Eval accuracy at epoch 4:  0.48853335
Train accuracy at epoch 5:  0.5000471
Eval accuracy at epoch 5:  0.5072
Train accuracy at epoch 6:  0.5000471
Eval accuracy at epoch 6:  0.4976
Train accuracy at epoch 7:  0.5209412
Eval accuracy at epoch 7:  0.5104
Train accuracy at epoch 8:  0.5130353
Eval accuracy at epoch 8:  0.51893336
Train accuracy at epoch 9:  0.50235295
Eval accuracy at epoch 9:  0.50346667
Train accuracy at epoch 10:  0.51981175
Eval accuracy at epoch 10:  0.5146667
Train accuracy at epoch 11:  0.59091765
Eval accuracy at epoch 11:  0.6002667
Train accuracy at epoch 12:  0.58174115
Eval accuracy at epoch 12:  0.59066665
Train accuracy at epoch 13:  0.6030588
Eval accuracy at epoch 13:  0.5976
Train accuracy at epoch 14:  0.54771763
Eval accuracy at epoch 14:  0.5477333
Train accuracy at epoch 15:  0.5787294
Eval accuracy at epoch 15:  0.58933336
Train accuracy at epoch 16:  0.55651766
Eval accuracy at epoch 16:  0.5648
Test accuracy:  0.5558
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicRNNCell
object at 0x7fec291f6c10>: Note that this cell is not optimized for pe
rformance. Please use tf.contrib.cudnn_rnn.CudnnRNNTanh for better per
formance on GPU.

/tmp/ipykernel_7766/2071740490.py:48: UserWarning: `tf.nn.rnn_cell.Bas
icRNNCell` is deprecated and will be removed in a future version. This
class is equivalent as `tf.keras.layers.SimpleRNNCell`, and will be re
placed by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicRNNCell(cell_size)
```

```
Train accuracy at epoch 1:  0.49962354
Eval accuracy at epoch 1:  0.5056
Train accuracy at epoch 2:  0.5215529
Eval accuracy at epoch 2:  0.5248
Train accuracy at epoch 3:  0.5579294
Eval accuracy at epoch 3:  0.56
Train accuracy at epoch 4:  0.6227294
Eval accuracy at epoch 4:  0.6333333
Train accuracy at epoch 5:  0.64395297
Eval accuracy at epoch 5:  0.644
Train accuracy at epoch 6:  0.65345883
Eval accuracy at epoch 6:  0.67013335
Train accuracy at epoch 7:  0.50042355
Eval accuracy at epoch 7:  0.51093334
Train accuracy at epoch 8:  0.50178826
Eval accuracy at epoch 8:  0.48666668
Train accuracy at epoch 9:  0.5019765
Eval accuracy at epoch 9:  0.50186664
Train accuracy at epoch 10:  0.5045647
Eval accuracy at epoch 10:  0.508
Train accuracy at epoch 11:  0.5016
Eval accuracy at epoch 11:  0.5016
Test accuracy:  0.50148
```





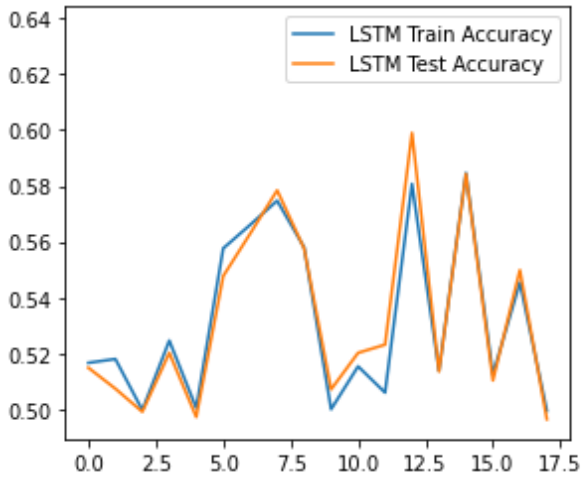
In [19]:

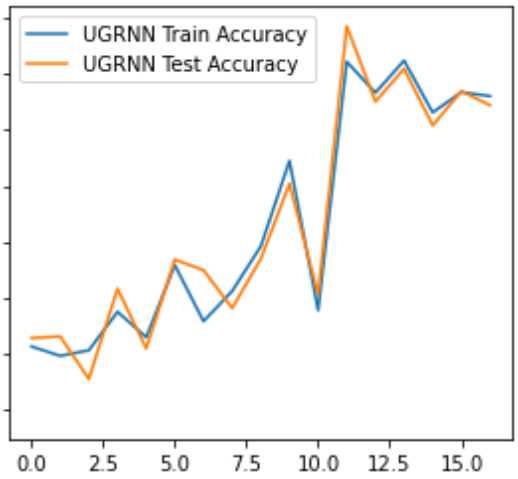
```
# Specify the path where you want to save/restore the trained variables.
checkpoint_directory = './models_checkpoints/ImdbRNN4/'

# Use the GPU if available.
device = 'gpu:0'

# Define optimizer.
optimizer = tf.compat.v1.train.GradientDescentOptimizer(learning_rate=1e-4)

# Instantiate model. This doesn't initialize the variables yet.
lstm_model4 = RNNModel(vocabulary_size=len(word2idx), device=device,
                       checkpoint_directory=checkpoint_directory)

# Train model
lstm_model4.fit(train_dataset, val_dataset, test_dataset, optimizer, num_epochs=20
,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(lstm_model4)
save_path = checkpoint.save(checkpoint_directory)

################################################################################
############################

# Define optimizer.
optimizer = tf.compat.v1.train.GradientDescentOptimizer(learning_rate=1e-4)

# Instantiate model. This doesn't initialize the variables yet.
ugrnn_model4 = RNNModel(vocabulary_size=len(word2idx), rnn_cell='ugrnn',
                        device=device, checkpoint_directory=checkpoint_directory)

# Train model
ugrnn_model4.fit(train_dataset,  val_dataset, test_dataset, optimizer, num_epochs=
20,
                early_stopping_rounds=5, verbose=1, train_from_scratch=True)

#lstm_model.save_model()
checkpoint = tf.train.Checkpoint(ugrnn_model4)
save_path = checkpoint.save(checkpoint_directory)

################################################################################
############################

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
ax1.plot(range(len(lstm_model4.history['train_acc'])), lstm_model4.history['train_
acc'],
         label='LSTM Train Accuracy');
ax1.plot(range(len(lstm_model4.history['eval_acc'])), lstm_model4.history['eval_ac
c'],
         label='LSTM Test Accuracy');
ax2.plot(range(len(ugrnn_model4.history['train_acc'])), ugrnn_model4.history['trai
n_acc'],
```

```
         label='UGRNN Train Accuracy');
ax2.plot(range(len(ugrnn_model4.history['eval_acc'])), ugrnn_model4.history['eval_
acc'],
         label='UGRNN Test Accuracy');
ax1.legend();
ax2.legend();
```

```
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicLSTMCel
l object at 0x7fec291e5040>: Note that this cell is not optimized for
performance. Please use tf.contrib.cudnn_rnn.CudnnLSTM for better perf
ormance on GPU.

/tmp/ipykernel_7766/2071740490.py:45: UserWarning: `tf.nn.rnn_cell.Bas
icLSTMCell` is deprecated and will be removed in a future version. Thi
s class is equivalent as `tf.keras.layers.LSTMCell`, and will be repla
ced by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicLSTMCell(cell_size)

Train accuracy at epoch 1:  0.5167059
Eval accuracy at epoch 1:  0.51493335
Train accuracy at epoch 2:  0.5180706
Eval accuracy at epoch 2:  0.5074667
Train accuracy at epoch 3:  0.49981177
Eval accuracy at epoch 3:  0.4992
Train accuracy at epoch 4:  0.5246588
Eval accuracy at epoch 4:  0.52026665
Train accuracy at epoch 5:  0.5005647
Eval accuracy at epoch 5:  0.49733335
Train accuracy at epoch 6:  0.55764705
Eval accuracy at epoch 6:  0.5477333
Train accuracy at epoch 7:  0.5660706
Eval accuracy at epoch 7:  0.5629333
Train accuracy at epoch 8:  0.57468235
Eval accuracy at epoch 8:  0.5784
Train accuracy at epoch 9:  0.5579765
Eval accuracy at epoch 9:  0.5576
Train accuracy at epoch 10:  0.5000941
Eval accuracy at epoch 10:  0.5072
Train accuracy at epoch 11:  0.5154353
Eval accuracy at epoch 11:  0.52026665
Train accuracy at epoch 12:  0.5060706
Eval accuracy at epoch 12:  0.5232
Train accuracy at epoch 13:  0.5807059
Eval accuracy at epoch 13:  0.59893334
Train accuracy at epoch 14:  0.5139294
Eval accuracy at epoch 14:  0.5136
Train accuracy at epoch 15:  0.5845647
Eval accuracy at epoch 15:  0.58426666
Train accuracy at epoch 16:  0.51261175
Eval accuracy at epoch 16:  0.5104
Train accuracy at epoch 17:  0.54545885
Eval accuracy at epoch 17:  0.5498667
Train accuracy at epoch 18:  0.49957648
Eval accuracy at epoch 18:  0.49653333
Test accuracy:  0.5022
WARNING:tensorflow:<keras.layers.legacy_rnn.rnn_cell_impl.BasicRNNCell
object at 0x7fec29165a30>: Note that this cell is not optimized for pe
rformance. Please use tf.contrib.cudnn_rnn.CudnnRNNTanh for better per
formance on GPU.
```

```
/tmp/ipykernel_7766/2071740490.py:48: UserWarning: `tf.nn.rnn_cell.Bas
icRNNCell` is deprecated and will be removed in a future version. This
class is equivalent as `tf.keras.layers.SimpleRNNCell`, and will be re
placed by that in Tensorflow 2.0.
  self.rnn_cell = tf.compat.v1.nn.rnn_cell.BasicRNNCell(cell_size)

Train accuracy at epoch 1:  0.52254117
Eval accuracy at epoch 1:  0.5256
Train accuracy at epoch 2:  0.5192
Eval accuracy at epoch 2:  0.52613336
Train accuracy at epoch 3:  0.52117646
Eval accuracy at epoch 3:  0.51093334
Train accuracy at epoch 4:  0.5349647
Eval accuracy at epoch 4:  0.5432
Train accuracy at epoch 5:  0.5258353
Eval accuracy at epoch 5:  0.5218667
Train accuracy at epoch 6:  0.5516235
Eval accuracy at epoch 6:  0.5536
Train accuracy at epoch 7:  0.53157645
Eval accuracy at epoch 7:  0.5498667
Train accuracy at epoch 8:  0.5423529
Eval accuracy at epoch 8:  0.5362667
Train accuracy at epoch 9:  0.5584
Eval accuracy at epoch 9:  0.5538667
Train accuracy at epoch 10:  0.58894116
Eval accuracy at epoch 10:  0.5808
Train accuracy at epoch 11:  0.53548235
Eval accuracy at epoch 11:  0.54106665
Train accuracy at epoch 12:  0.6243765
Eval accuracy at epoch 12:  0.63706666
Train accuracy at epoch 13:  0.61331767
Eval accuracy at epoch 13:  0.61013335
Train accuracy at epoch 14:  0.6248
Eval accuracy at epoch 14:  0.62186664
Train accuracy at epoch 15:  0.6062588
Eval accuracy at epoch 15:  0.6016
Train accuracy at epoch 16:  0.6135059
Eval accuracy at epoch 16:  0.6138667
Train accuracy at epoch 17:  0.6120941
Eval accuracy at epoch 17:  0.6088
Test accuracy:  0.59728
```





In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [11]:

```
####################################################################
# Import/download necessary libraries to process new sequences
####################################################################
import nltk
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')
from nltk.tokenize import word_tokenize
import re
```

In [12]:

```
def process_new_review(review):
    '''Function to process a new review.
       Args:
           review: original text review, string.
       Returns:
           indexed_review: sequence of integers, words correspondence
                           from word2idx.
           seq_length: the length of the review.
    '''
    indexed_review = re.sub(r'<[^>]+>', ' ', review)
    indexed_review = word_tokenize(indexed_review)
    indexed_review = [word2idx[word] if word in list(word2idx.keys()) else
                      word2idx['Unknown_token'] for word in indexed_review]
    indexed_review = indexed_review + [word2idx['End_token']]
    seq_length = len(indexed_review)
    return indexed_review, seq_length
```

In [13]:

```
sent_dict = {0: 'negative', 1: 'positive'}
```

In [14]:

```
review_score_10 = "I think Bad Apples is a great time and I recommend! I enjoyed t
he opening, which gave way for the rest of the movie to occur. The main couple was
very likable and I believed all of their interactions. They had great onscreen che
mistry and made me laugh quite a few times! Keeping the girls in the masks but see
ing them in action was something I loved. It kept a mystery to them throughout. I
 think the dialogue was great. The kills were fun. And the special surprise gore e
ffect at the end was AWESOME!! I won't spoil that part ;) I also enjoyed how the m
ovie wrapped up. It gave a very urban legends type feel of \"did you ever hear the
story...\". Plus is leaves the door open for another film which I wouldn't mind at
all. Long story short, I think if you take the film for what it is; a fun little h
orror flick, then you won't be disappointed! HaPpY eArLy HaLLoWeEn!"
```

In [15]:

```
review_score_4 = "A young couple comes to a small town, where the husband get a jo
b working in a hospital. The wife which you instantly hate or dislike works home,
 at the same time a horrible murders takes place in this small town by two masked
 killers. Bad Apples is just your tipical B-horror movie with average acting (I gi
ve them that. Altough you may get the idea that some of the actors are crazy-conve
rvative Christians), but the script is just bad, and that's what destroys the fil
m."
```

In [16]:

```
review_score_1 = "When you first start watching this movie, you can tell its going
to be a painful ride. the audio is poor...the attacks by the \"girls\" are like go
ing back in time, to watching the old rocky films, were blows never touched. the e
diting is poor with it aswell, example the actress in is the bath when her husband
comes home, clearly you see her wearing a flesh coloured bra in the bath. no hints
or spoilers, just wait till you find it in a bargain basket of cheap dvds in a cou
ple of weeks"
```

In [17]:

```
new_reviews = [review_score_10, review_score_4, review_score_1]
scores = [10, 4, 1]
```

In [18]:

```
with tf.device(device):
    for original_review, score in zip(new_reviews, scores):
        indexed_review, seq_length = process_new_review(original_review)
        indexed_review = tf.reshape(tf.constant(indexed_review), (1,-1))
        seq_length = tf.reshape(tf.constant(seq_length), (1,))
        logits = lstm_model.predict(indexed_review, seq_length, False)
        pred = tf.argmax(logits, axis=1).numpy()[0]
        print('The sentiment for the review with score %d was found to be %s'
              %(score, sent_dict[pred]))
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
/tmp/ipykernel_7766/413355369.py in <module>
      4         indexed_review = tf.reshape(tf.constant(indexed_review
), (1,-1))
      5         seq_length = tf.reshape(tf.constant(seq_length), (1,))
----> 6         logits = lstm_model.predict(indexed_review, seq_length
, False)
      7         pred = tf.argmax(logits, axis=1).numpy()[0]
      8         print('The sentiment for the review with score %d was
 found to be %s'

NameError: name 'lstm_model' is not defined
```