In [7]:
```
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

import sqlite3
import datetime as dt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from itertools import islice

from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.preprocessing import sequence
from keras.initializers import he_normal
from keras.layers import BatchNormalization, Dense, Dropout, Flatten, LSTM
from keras.layers.embeddings import Embedding
from keras.regularizers import L1L2
```

```
Using TensorFlow backend.
```

In [8]:
```
connection = sqlite3.connect('final.sqlite')
# Load data into pandas dataframe.
reviews_df = pd.read_sql_query(""" SELECT * FROM Reviews """,connection)

    # Drop index column
reviews_df = reviews_df.drop(columns=['index'])
reviews_df['Time'] = reviews_df[['Time']].applymap(lambda x: dt.datetime.fromtimest
amp(x))


reviews_df=reviews_df.sample(50000)

    # Sort the data on the basis of time.
reviews_df = reviews_df.sort_values(by=['Time'])
cleaned_text = reviews_df['CleanedText'].values

print("Dataset Shape : \n",cleaned_text.shape)
```

```
Dataset Shape :
 (50000,)
```

In [9]:
```
all_words=[]
for sentence in cleaned_text:
    words = sentence.split()
    all_words += words

print("Shape of the data : ",cleaned_text.shape)
print("Number of sentences present in complete dataset : ",len(all_words))

counts = Counter(all_words)
print("Number of unique words present in whole corpus: ",len(counts.most_common()))
vocab_size = len(counts.most_common()) + 1
top_words_count = 5000
sorted_words = counts.most_common(top_words_count)

word_index_lookup = dict()
i = 1
for word,frequency in sorted_words:
    word_index_lookup[word] = i
    i += 1

print()
print("Top 25 words with their frequencies:")
print(counts.most_common(25))
print()
print("Top 25 words with their index:")
print(list(islice(word_index_lookup.items(), 25)))
```

```
Shape of the data :  (50000,)
Number of sentences present in complete dataset :  1793783
Number of unique words present in whole corpus:  27002

Top 25 words with their frequencies:
[('like', 21033), ('tast', 21020), ('tea', 17974), ('good', 16777), ('flavor', 1
6771), ('great', 15565), ('product', 14812), ('one', 14672), ('love', 14609), ('
use', 14514), ('tri', 12949), ('make', 11617), ('get', 10395), ('coffe', 8748),
('food', 8308), ('eat', 8283), ('would', 8206), ('buy', 7973), ('time', 7922), (
'best', 7750), ('realli', 7547), ('also', 7425), ('find', 7365), ('dont', 7168),
('amazon', 7096)]

Top 25 words with their index:
[('like', 1), ('tast', 2), ('tea', 3), ('good', 4), ('flavor', 5), ('great', 6),
('product', 7), ('one', 8), ('love', 9), ('use', 10), ('tri', 11), ('make', 12),
('get', 13), ('coffe', 14), ('food', 15), ('eat', 16), ('would', 17), ('buy', 18
), ('time', 19), ('best', 20), ('realli', 21), ('also', 22), ('find', 23), ('don
t', 24), ('amazon', 25)]
```

In [10]:
```
def apply_text_index(row):
    holder = []
    for word in row['CleanedText'].split():
        if word in word_index_lookup:
            holder.append(word_index_lookup[word])
        else:
            holder.append(0)
    return holder


reviews_df['CleanedText_Index'] = reviews_df.apply(lambda row: apply_text_index(row
),axis=1)
reviews_df.head(5)
```

Out[10]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	S
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	pos
30	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	pos
424	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	pos
330	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	pos
423	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	pos

In [11]:
```
reviews_df['Score'] = reviews_df['Score'].map(lambda x : 1 if x == 'positive' else
0)
reviews_df.head(5)
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Sc
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	
30	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	
424	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	
330	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	
423	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	

In [12]:
```
x_train, x_test, y_train, y_test = train_test_split(reviews_df['CleanedText_Index']
.values,
                                                    reviews_df['Score'],
                                                    test_size=0.3,
                                                    shuffle=False,
                                                    random_state=0)
```

In [12]:
```
x_train, x_test, y_train, y_test = train_test_split(reviews_df['CleanedText_Index']
.values,
                                                    reviews_df['Score'],
                                                    test_size=0.3,
                                                    shuffle=False,
                                                    random_state=0)
```

In [13]:
```
print("Total number words present in first review:\n",len(x_train[1]))
print()
print("List of word indexes present in first review:\n", x_train[1])
print()
```

```
Total number words present in first review:
 30

List of word indexes present in first review:
 [499, 158, 781, 1094, 0, 43, 323, 836, 1146, 586, 74, 45, 3385, 122, 10, 2759,
797, 3581, 1963, 2869, 4411, 343, 116, 856, 76, 523, 797, 615, 561, 251]
```

In [14]:
```
max_review_length = 500
x_train = sequence.pad_sequences(x_train, maxlen=max_review_length)
x_test = sequence.pad_sequences(x_test, maxlen=max_review_length)

print("Total number words present in first review after padding:\n",len(x_train[1])
)
print()
print("List of word indexes present in first review padding:\n", x_train[1])
print()
```

```
Total number words present in first review after padding:
 500

List of word indexes present in first review padding:
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0  499  158  781 1094    0   43
  323  836 1146  586   74   45 3385  122   10 2759  797 3581 1963 2869
 4411  343  116  856   76  523  797  615  561  251]
```

1 LSTM Layer

In [15]:
```
batch_size = 192

# Number of time whole data is trained
epochs = 10

# Embedding vector size
embedding_vecor_length = 32

# Bias regularizer value - we will use elasticnet
reg = L1L2(0.01, 0.01)

# Plot train and cross validation loss
def plot_train_cv_loss(trained_model, epochs, colors=['b']):
    fig, ax = plt.subplots(1,1)
    ax.set_xlabel('epoch')
    ax.set_ylabel('Categorical Crossentropy Loss')
    x_axis_values = list(range(1,epochs+1))

    validation_loss = trained_model.history['val_loss']
    train_loss = trained_model.history['loss']

    ax.plot(x_axis_values, validation_loss, 'b', label="Validation Loss")
    ax.plot(x_axis_values, train_loss, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

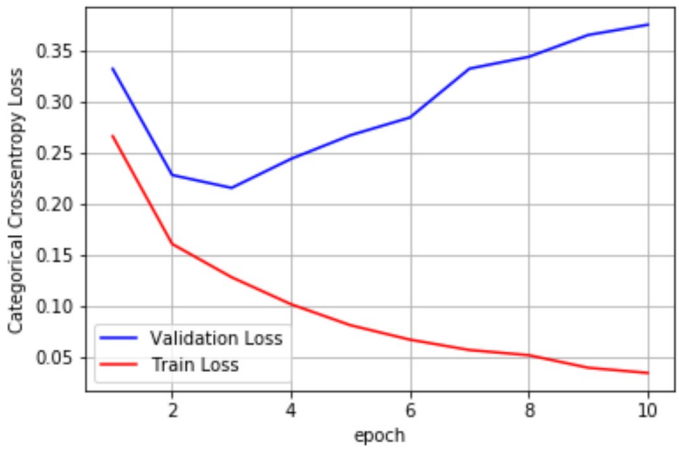
In [16]:
```
model = Sequential()

# Add Embedding Layer
model.add(Embedding(vocab_size, embedding_vecor_length, input_length=max_review_len
gth))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer
model.add(LSTM(100))

# Add dropout
model.add(Dropout(0.20))

# Add Dense Layer
model.add(Dense(1, activation='sigmoid'))

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, np.array(y_train), batch_size = batch_size, epoc
hs = epochs, verbose=1, validation_data=(x_test, y_test))
```

```
Model Summary:

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           864096
_________________________________________________________________
batch_normalization_1 (Batch (None, 500, 32)           128
_________________________________________________________________
dropout_1 (Dropout)          (None, 500, 32)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 917,525
Trainable params: 917,461
Non-trainable params: 64
_________________________________________________________________


Train on 35000 samples, validate on 15000 samples
Epoch 1/10
35000/35000 [==============================] - 337s 10ms/step - loss: 0.2658 - a
cc: 0.9049 - val_loss: 0.3315 - val_acc: 0.8988
Epoch 2/10
35000/35000 [==============================] - 315s 9ms/step - loss: 0.1607 - ac
c: 0.9401 - val_loss: 0.2279 - val_acc: 0.9227
Epoch 3/10
35000/35000 [==============================] - 357s 10ms/step - loss: 0.1282 - a
cc: 0.9534 - val_loss: 0.2154 - val_acc: 0.9224
Epoch 4/10
35000/35000 [==============================] - 365s 10ms/step - loss: 0.1018 - a
cc: 0.9640 - val_loss: 0.2437 - val_acc: 0.9215
Epoch 5/10
35000/35000 [==============================] - 396s 11ms/step - loss: 0.0813 - a
cc: 0.9717 - val_loss: 0.2668 - val_acc: 0.9197
Epoch 6/10
35000/35000 [==============================] - 412s 12ms/step - loss: 0.0673 - a
cc: 0.9769 - val_loss: 0.2840 - val_acc: 0.9187
Epoch 7/10
35000/35000 [==============================] - 633s 18ms/step - loss: 0.0572 - a
cc: 0.9803 - val_loss: 0.3317 - val_acc: 0.9186
Epoch 8/10
35000/35000 [==============================] - 836s 24ms/step - loss: 0.0521 - a
cc: 0.9824 - val_loss: 0.3433 - val_acc: 0.9150
Epoch 9/10
35000/35000 [==============================] - 939s 27ms/step - loss: 0.0399 - a
cc: 0.9863 - val_loss: 0.3646 - val_acc: 0.9120
Epoch 10/10
35000/35000 [==============================] - 655s 19ms/step - loss: 0.0348 - a
cc: 0.9880 - val_loss: 0.3746 - val_acc: 0.9116
```

In [17]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy: {0:.2f}%'.format(score[1]*100))
```

```
Test accuracy: 91.16%
```

In [18]:
```
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```



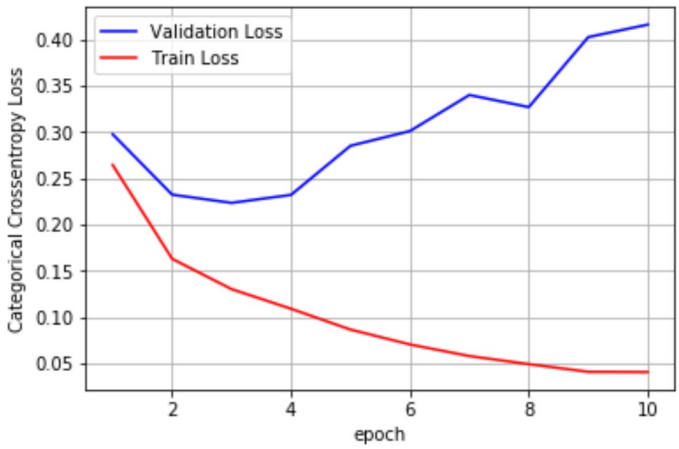
2 LSTM Layer

In [19]:
```
%%time
# Instantiate sequntial model
model = Sequential()

# Add Embedding Layer
model.add(Embedding(vocab_size, embedding_vecor_length, input_length=max_review_len
gth))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 1
model.add(LSTM(100,return_sequences=True))

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 2
model.add(LSTM(100))

# Add dropout
model.add(Dropout(0.20))

# Add Dense Layer
model.add(Dense(1, activation='sigmoid'))

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, np.array(y_train), batch_size = batch_size, epoc
hs = epochs, verbose=1, validation_data=(x_test, y_test))
```

```
Model Summary:

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           864096
_________________________________________________________________
batch_normalization_2 (Batch (None, 500, 32)           128
_________________________________________________________________
dropout_3 (Dropout)          (None, 500, 32)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 500, 100)          53200
_________________________________________________________________
dropout_4 (Dropout)          (None, 500, 100)          0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout_5 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 997,925
Trainable params: 997,861
Non-trainable params: 64
_________________________________________________________________


Train on 35000 samples, validate on 15000 samples
Epoch 1/10
35000/35000 [==============================] - 1476s 42ms/step - loss: 0.2645 -
acc: 0.9071 - val_loss: 0.2976 - val_acc: 0.9019
Epoch 2/10
35000/35000 [==============================] - 818s 23ms/step - loss: 0.1631 - a
cc: 0.9389 - val_loss: 0.2325 - val_acc: 0.9189
Epoch 3/10
35000/35000 [==============================] - 818s 23ms/step - loss: 0.1304 - a
cc: 0.9522 - val_loss: 0.2236 - val_acc: 0.9256
Epoch 4/10
35000/35000 [==============================] - 914s 26ms/step - loss: 0.1092 - a
cc: 0.9608 - val_loss: 0.2322 - val_acc: 0.9224
Epoch 5/10
35000/35000 [==============================] - 824s 24ms/step - loss: 0.0868 - a
cc: 0.9707 - val_loss: 0.2852 - val_acc: 0.9205
Epoch 6/10
35000/35000 [==============================] - 756s 22ms/step - loss: 0.0707 - a
cc: 0.9757 - val_loss: 0.3011 - val_acc: 0.9205
Epoch 7/10
35000/35000 [==============================] - 781s 22ms/step - loss: 0.0582 - a
cc: 0.9805 - val_loss: 0.3401 - val_acc: 0.9164
Epoch 8/10
35000/35000 [==============================] - 782s 22ms/step - loss: 0.0494 - a
cc: 0.9833 - val_loss: 0.3270 - val_acc: 0.9146
Epoch 9/10
35000/35000 [==============================] - 755s 22ms/step - loss: 0.0412 - a
cc: 0.9856 - val_loss: 0.4026 - val_acc: 0.9179
Epoch 10/10
35000/35000 [==============================] - 1213s 35ms/step - loss: 0.0409 -
acc: 0.9860 - val_loss: 0.4161 - val_acc: 0.9126
Wall time: 2h 32min 27s
```

In [20]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy: {0:.2f}%'.format(score[1]*100))
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```

```
Test accuracy: 91.26%
```



3 LSTM Layer

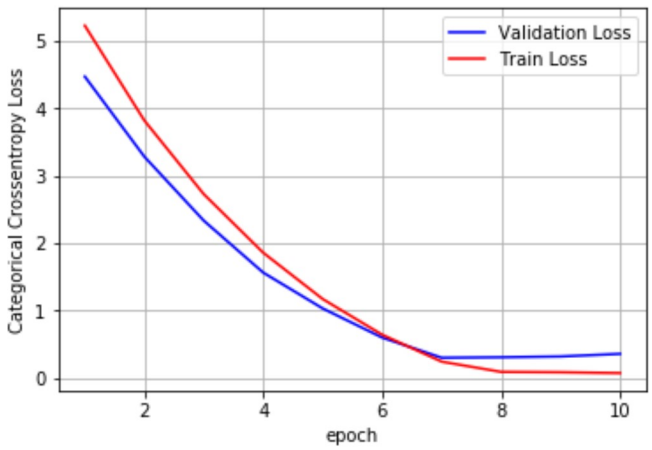
In [21]:
```
model = Sequential()

# Add Embedding Layer
model.add(Embedding(vocab_size, embedding_vecor_length, input_length=max_review_len
gth))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 1
model.add(LSTM(100,return_sequences=True,bias_regularizer=reg))

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 2
model.add(LSTM(80,return_sequences=True,bias_regularizer=reg))

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 3
model.add(LSTM(60,return_sequences=True,bias_regularizer=reg))

# Add dropout
model.add(Dropout(0.30))

# Add LSTM Layer 4
model.add(LSTM(40,return_sequences=True,bias_regularizer=reg))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.40))

# Add LSTM Layer 5
model.add(LSTM(20))

# Add dropout
model.add(Dropout(0.50))

# Add Dense Layer
model.add(Dense(1, activation='sigmoid'))

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, np.array(y_train), batch_size = batch_size, epoc
hs = epochs, verbose=1, validation_data=(x_test, y_test))
```

```
Model Summary:

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 500, 32)           864096
_________________________________________________________________
batch_normalization_3 (Batch (None, 500, 32)           128
_________________________________________________________________
dropout_6 (Dropout)          (None, 500, 32)           0
_________________________________________________________________
lstm_4 (LSTM)                (None, 500, 100)          53200
_________________________________________________________________
dropout_7 (Dropout)          (None, 500, 100)          0
_________________________________________________________________
lstm_5 (LSTM)                (None, 500, 80)           57920
_________________________________________________________________
dropout_8 (Dropout)          (None, 500, 80)           0
_________________________________________________________________
lstm_6 (LSTM)                (None, 500, 60)           33840
_________________________________________________________________
dropout_9 (Dropout)          (None, 500, 60)           0
_________________________________________________________________
lstm_7 (LSTM)                (None, 500, 40)           16160
_________________________________________________________________
batch_normalization_4 (Batch (None, 500, 40)           160
_________________________________________________________________
dropout_10 (Dropout)         (None, 500, 40)           0
_________________________________________________________________
lstm_8 (LSTM)                (None, 20)                4880
_________________________________________________________________
dropout_11 (Dropout)         (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 21
=================================================================
Total params: 1,030,405
Trainable params: 1,030,261
Non-trainable params: 144
_________________________________________________________________


Train on 35000 samples, validate on 15000 samples
Epoch 1/10
35000/35000 [==============================] - 3090s 88ms/step - loss: 5.2278 -
acc: 0.8877 - val_loss: 4.4712 - val_acc: 0.8883
Epoch 2/10
35000/35000 [==============================] - 2180s 62ms/step - loss: 3.8200 -
acc: 0.9122 - val_loss: 3.2830 - val_acc: 0.9165
Epoch 3/10
35000/35000 [==============================] - 1937s 55ms/step - loss: 2.7252 -
acc: 0.9354 - val_loss: 2.3321 - val_acc: 0.9216
Epoch 4/10
35000/35000 [==============================] - 1936s 55ms/step - loss: 1.8562 -
acc: 0.9475 - val_loss: 1.5607 - val_acc: 0.9215
Epoch 5/10
35000/35000 [==============================] - 2827s 81ms/step - loss: 1.1682 -
acc: 0.9573 - val_loss: 1.0287 - val_acc: 0.9203
Epoch 6/10
35000/35000 [==============================] - 2600s 74ms/step - loss: 0.6386 -
acc: 0.9641 - val_loss: 0.5964 - val_acc: 0.9198
Epoch 7/10
35000/35000 [==============================] - 1940s 55ms/step - loss: 0.2390 -
acc: 0.9685 - val_loss: 0.2973 - val_acc: 0.9138
Epoch 8/10
```

In [22]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy: {0:.2f}%'.format(score[1]*100))
```

```
Test accuracy: 91.33%
```

In [23]:
```
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```



In [ ]: