Import Library for Data Processing

In [114]:

```
!pip install PySastrawi
```

```
Requirement already satisfied: PySastrawi in /usr/local/lib/python3.7/dist-packages (1.2.
0)
```

In [115]:

```
import numpy as np # berguna untuk rumus matematika
import pandas as pd # berguna untuk perdataan
```

Read CSV File

In [116]:

```
df = pd.read_csv('data.csv', encoding='latin-1')

alay_dict = pd.read_csv('new_kamusalay.csv', encoding='latin-1', header=None)
alay_dict = alay_dict.rename(columns={0: 'original',
                                      1: 'replacement'})

id_stopword_dict = pd.read_csv('stopwordbahasa.csv', header=None)
id_stopword_dict = id_stopword_dict.rename(columns={0: 'stopword'})
```

In [117]:

```
print("Shape: ", df.shape)
df.head(15)
```

```
Shape:  (13169, 13)
```

Out[117]:

	Tweet	HS	Abusive	HS_Individual	HS_Group	HS_Religion	HS_Race	HS_Physical	HS_Gender	HS_Other
0	- disaat semua cowok berusaha melacak perhatia...	1	1	1	0	0	0	0	0	1
1	RT USER: USER siapa yang telat ngasih tau elu?...	0	1	0	0	0	0	0	0	0
2	41. Kadang aku berfikir, kenapa aku tetap perc...	0	0	0	0	0	0	0	0	0
3	USER USER AKU ITU AKU\n\nKU TAU MATAMU SIPIT T...	0	0	0	0	0	0	0	0	0
4	USER USER Kaum cebong kapir udah keliatan dong...	1	1	0	1	1	0	0	0	0
5	USER Ya bani taplak dkk \xf0\x9f\x98\x84\xf0\x...	1	1	0	1	0	0	0	0	1
6	deklarasi pilkada 2018 aman dan anti hoax warg...	0	0	0	0	0	0	0	0	0
7	Gue baru aja kelar re-watch Aldnoah Zero!!! pa...	0	1	0	0	0	0	0	0	0
8	Nah admin belanja satu lagi port terbaik nak	0	0	0	0	0	0	0	0	0

	Tweet	HS	Abusive	HS_Individual	HS_Group	HS_Religion	HS_Race	HS_Physical	HS_Gender	HS_Othe
9	USER Enak lg klo smbil ngewe'	0	1	0	0	0	0	0	0	0
10	Setidaknya gw punya jari tengah buat lu, sebel...	1	1	1	0	0	0	0	0	1
11	USER USER USER USER BANCI KALENG MALU GA BISA ...	1	1	1	0	0	0	0	1	0
12	Kalo belajar ekonomi mestinya jago memprivatis...	0	0	0	0	0	0	0	0	0
13	Aktor huruhara 98 Prabowo S ingin lengserkan p...	1	0	1	0	0	0	0	0	1
14	USER Bu guru enakan jadi jablay atau guru esde...	1	1	1	0	0	0	0	1	0

Count Each Category

In [118]:

```
df.Abusive.value_counts()
```

Out[118]:

```
0    8126
1    5043
Name: Abusive, dtype: int64
```

In [119]:

```
print("Toxic shape: ", df[(df['Abusive'] == 1)].shape)
print("Non-toxic shape: ", df[(df['Abusive'] == 0)].shape)
```

```
Toxic shape:  (5043, 13)
Non-toxic shape:  (8126, 13)
```

Alay Dict

In [120]:

```
print("Shape: ", alay_dict.shape)
alay_dict.head(15)
```

```
Shape:  (15167, 2)
```

Out[120]:

	original	replacement
0	anakjakartaasikasik	anak jakarta asyik asyik
1	pakcikdahtua	pak cik sudah tua
2	pakcikmudalagi	pak cik muda lagi
3	t3tapjokowi	tetap jokowi
4	3x	tiga kali
5	aamiin	amin
6	aamiinn	amin
7	aamin	amin
8	aammiin	amin

	original	replacement
9	abis	habis
10	abisin	habiskan
11	acau	kacau
12	achok	ahok
13	ad	ada
14	adek	adik

ID Stopword

In [121]:

```
print("Shape: ", id_stopword_dict.shape)
id_stopword_dict.head()
```

```
Shape:  (758, 1)
```

Out[121]:

	stopword
0	ada
1	adalah
2	adanya
3	adapun
4	agak

Data Preprocess

In [122]:

```
import re
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
factory = StemmerFactory()
stemmer = factory.create_stemmer()

def lowercase(text):
    return text.lower()

def remove_unnecessary_char(text):
    text = re.sub('\n',' ',text) # Remove every '\n'
    text = re.sub('rt',' ',text) # Remove every retweet symbol
    text = re.sub('user',' ',text) # Remove every username
    text = re.sub('((www\.[^\s]+)|(https?://[^\s]+)|(http?://[^\s]+))',' ',text) # Remove
every URL
    text = re.sub('  +', ' ', text) # Remove extra spaces
    return text

def remove_nonaplhanumeric(text):
    text = re.sub('[^0-9a-zA-Z]+', ' ', text)
    return text

alay_dict_map = dict(zip(alay_dict['original'], alay_dict['replacement']))
def normalize_alay(text):
    return ' '.join([alay_dict_map[word] if word in alay_dict_map else word for word in
text.split(' ')])

def remove_stopword(text):
    text = ' '.join(['' if word in id_stopword_dict.stopword.values else word for word i
n text.split(' ')])
    text = re.sub('  +', ' ', text) # Remove extra spaces
    text = text.strip()
    return text

def stemming(text):
    return stemmer.stem(text)

print("remove_nonaplhanumeric: ", remove_nonaplhanumeric("Halooo,,,,, duniaa!!"))
print("lowercase: ", lowercase("Halooo, duniaa!"))
print("stemming: ", stemming("Perekonomian Indonesia sedang dalam pertumbuhan yang memban
ggakan"))
print("remove_unnecessary_char: ", remove_unnecessary_char("Hehe\n\n RT USER USER apa kab
s www.google.com\n  hehe"))
print("normalize_alay: ", normalize_alay("aamiin adek abis"))
print("remove_stopword: ", remove_stopword("ada hehe adalah huhu yang hehe"))
```

```
remove_nonaplhanumeric:  Halooo duniaa
lowercase:  halooo, duniaa!
stemming:  ekonomi indonesia sedang dalam tumbuh yang bangga
remove_unnecessary_char:  Hehe RT USER USER apa kabs hehe
normalize_alay:  amin adik habis
remove_stopword:  hehe huhu hehe
```

In [123]:

```
def preprocess(text):
    text = lowercase(text) # 1
    text = remove_nonaplhanumeric(text) # 2
    text = remove_unnecessary_char(text) # 3
    text = normalize_alay(text) # 4
    text = stemming(text) # 5
    text = remove_stopword(text) # 6
    return text
```

In [124]:

```
df['Tweet'] = df['Tweet'].apply(preprocess)
```

In [125]:

```
print("Shape: ", df.shape)
df.head(15)
```

```
Shape:  (13169, 13)
```

Out[125]:

	Tweet	HS	Abusive	HS_Individual	HS_Group	HS_Religion	HS_Race	HS_Physical	HS_Gender	HS_Other	HS_Weak
0	cowok usaha lacak perhati gue lantas remeh per...	1	1	1	0	0	0	0	0	1	1
1	telat tau edan sarap gue gaul cigax jifla cal ...	0	1	0	0	0	0	0	0	0	0
2	41 kadang pikir percaya tuhan jatuh kali kali ...	0	0	0	0	0	0	0	0	0	0
3	ku tau mata sipit lihat	0	0	0	0	0	0	0	0	0	0
4	kaum cebong kafir lihat dongok	1	1	0	1	1	0	0	0	0	0

	Tweet	HS	Abusive	HS_Individual	HS_Group	HS_Religion	HS_Race	HS_Physical	HS_Gender	HS_Other	HS_Weak
	dungu haha										
5	ya bani taplak kawan kawan xf0 x9f x98 x84 xf0...	1	1	0	1	0	0	0	0	1	0
6	deklarasi pilih kepala daerah 2018 aman anti h...	0	0	0	0	0	0	0	0	0	0
7	gue selesai re watch aldnoah zero kampret 2 ka...	0	1	0	0	0	0	0	0	0	0
8	admin belanja po nak makan ais kepal milo ais ...	0	0	0	0	0	0	0	0	0	0
9	enak ngewe	0	1	0	0	0	0	0	0	0	0
10	gue jari gue ukur nyali bacot xf0 x9f x98 x8f	1	1	1	0	0	0	0	0	1	1
11	banci kaleng malu pe anyaan 2 nyungsep koe uni...	1	1	1	0	0	0	0	1	0	1
12	ajar ekonomi mesti jago privatisasi hati orang...	0	0	0	0	0	0	0	0	0	0
13	aktor huru hara 98 prabowo si lengser perintah...	1	0	1	0	0	0	0	0	1	1
14	bu guru enak jablay guru sekolah dasar sih kay...	1	1	1	0	0	0	0	1	0	1

In [126]:

```
# data.to_csv('tweet.csv', index=False)
```

Import Library for Learning

In [127]:

```
import os
import tweepy as tw


#For Preprocessing
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
from nltk.stem.porter import *

# For Building the model
from sklearn.model_selection import train_test_split
import tensorflow as tf
import seaborn as sns

#For data visualization
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [128]:

```
# Check for missing data
df.isnull().sum()
```

Out[128]:

```
Tweet            0
HS               0
Abusive          0
HS_Individual    0
HS_Group         0
HS_Religion      0
HS_Race          0
HS_Physical      0
HS_Gender        0
HS_Other         0
HS_Weak          0
HS_Moderate      0
HS_Strong        0
dtype: int64
```

In [129]:

```
# dimensionality of the data
df.shape
```

Out[129]:

```
(13169, 13)
```

Drop Missing Rows

In [130]:

```
# drop missing rows
df.dropna(axis=0, inplace=True)
```

Plotting the Pie chart of the percentage of different sentiments of all the tweets

In [131]:

```
print("Toxic shape: ", df[(df['Abusive'] == 1)].shape)
print("Non-toxic shape: ", df[(df['Abusive'] == 0)].shape)
```

```
Toxic shape:  (5043, 13)
Non-toxic shape:  (8126, 13)
```

In [132]:

```
import plotly.express as px
fig = px.pie(df, names='Abusive', title ='Pie chart of different Abusive or Not of tweets
')
fig.show()
```

Data Preprocessing for LSTM Learning

In [133]:

```
def tweet_to_words(tweet):
    ''' Convert tweet text into a sequence of words '''

    # convert to lowercase
    text = tweet.lower()
    # tokenize
    words = text.split()
    # return list
    return words

print("\nOriginal tweet ->", df['Tweet'][0])
print("\nProcessed tweet ->", tweet_to_words(df['Tweet'][0]))
```

```
Original tweet -> cowok usaha lacak perhati gue lantas remeh perhati gue kasih khusus bas
ic cowok bego

Processed tweet -> ['cowok', 'usaha', 'lacak', 'perhati', 'gue', 'lantas', 'remeh', 'perh
ati', 'gue', 'kasih', 'khusus', 'basic', 'cowok', 'bego']
```

In [134]:

```
# Apply data processing to each tweet
X = list(map(tweet_to_words, df['Tweet']))
```

In [135]:

```
from sklearn.preprocessing import LabelEncoder

# Encode target labels
le = LabelEncoder()
Y = le.fit_transform(df['Abusive'])
```

In [136]:

```
print(X[0])
print(Y[0])
```

```
['cowok', 'usaha', 'lacak', 'perhati', 'gue', 'lantas', 'remeh', 'perhati', 'gue', 'kasih
', 'khusus', 'basic', 'cowok', 'bego']
1
```

Train and test split

In [137]:

```
y = pd.get_dummies(df['Abusive'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, rando
m_state=1)
```

Bag of words (BOW) feature extraction

In [138]:

```
from sklearn.feature_extraction.text import CountVectorizer
#from sklearn.feature_extraction.text import TfidfVectorizer

vocabulary_size = 5000

# Tweets have already been preprocessed hence dummy function will be passed in
# to preprocessor & tokenizer step
count_vector = CountVectorizer(max_features=vocabulary_size,
#                               ngram_range=(1,2),    # unigram and bigram
                                preprocessor=lambda x: x,
                               tokenizer=lambda x: x)
#tfidf_vector = TfidfVectorizer(lowercase=True, stop_words='english')

# Fit the training data
X_train = count_vector.fit_transform(X_train).toarray()

# Transform testing data
X_test = count_vector.transform(X_test).toarray()
```

In [139]:

```
import sklearn.preprocessing as pr

# Normalize BoW features in training and test set
X_train = pr.normalize(X_train, axis=1)
X_test  = pr.normalize(X_test, axis=1)
```

Tokenizing & Padding

In [140]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

max_words = 5000
max_len=50

def tokenize_pad_sequences(text):
    '''
    This function tokenize the input text into sequnences of intergers and then
    pad each sequence to the same length
    '''
    # Text tokenization
    tokenizer = Tokenizer(num_words=max_words, lower=True, split=' ')
    tokenizer.fit_on_texts(text)
    # Transforms text to a sequence of integers
    X = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    X = pad_sequences(X, padding='post', maxlen=max_len)
    # return sequences
    return X, tokenizer

print('Before Tokenization & Padding \n', df['Tweet'][0])
X, tokenizer = tokenize_pad_sequences(df['Tweet'])
print('After Tokenization & Padding \n', X[0])
```

```
Before Tokenization & Padding
 cowok usaha lacak perhati gue lantas remeh perhati gue kasih khusus basic cowok bego
After Tokenization & Padding
 [ 324  161 3546  608    7 2017 2575  608    7   83  127  324  209    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0]
```

In [141]:

```
print('Before Tokenization & Padding \n', df['Tweet'][300])
X, tokenizer = tokenize_pad_sequences(df['Tweet'])
print('After Tokenization & Padding \n', X[300])
```

```
Before Tokenization & Padding
 bubar hati amp front bela islam uniform resource locator
After Tokenization & Padding
 [121  66  55 702 140  10  12  13  14   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
```

Saving tokenized data

In [142]:

```
import pickle

# saving
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

# loading
with open('tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)
```

80 20 ratio

Performing learning for 80% data training and 20% data testing.

Train & Test Split

In [143]:

```
y = pd.get_dummies(df['Abusive'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, rando
m_state=1)
print('Train Set ->', X_train.shape, y_train.shape)
print('Validation Set ->', X_val.shape, y_val.shape)
print('Test Set ->', X_test.shape, y_test.shape)
```

```
Train Set -> (8428, 50) (8428, 2)
```

```
Validation Set -> (2107, 50) (2107, 2)
Test Set -> (2634, 50) (2634, 2)
```

In [144]:

```
import keras.backend as K

def f1_score(precision, recall):
    ''' Function to calculate f1 score '''

    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

Bidirectional LSTM Using NN

In [145]:

```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dr
opout
from keras.metrics import Precision, Recall
from keras import datasets

from keras.callbacks import LearningRateScheduler
from keras.callbacks import History

from keras import losses

vocab_size = 10000
embedding_size = 1100
epochs=10

# Build model
model= Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model.add(LSTM(64, activation="ReLU", return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(8, activation="ReLU"))
model.add(Dropout(0.2))
model.add(Dense(2, activation='softmax'))
```

In [146]:

```
import tensorflow
tensorflow.keras.utils.to_categorical(y_train)
tensorflow.keras.utils.to_categorical(y_test)
```

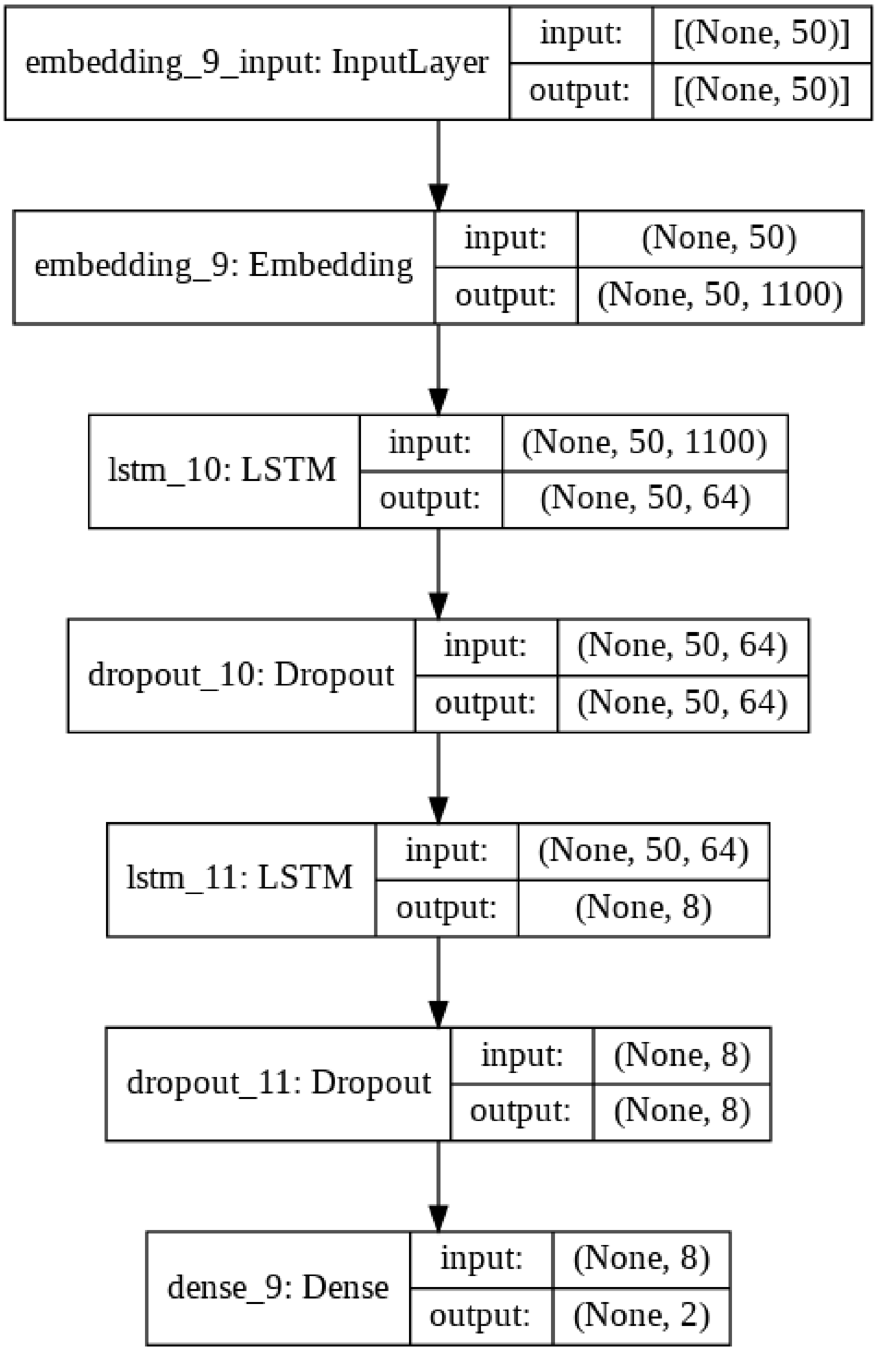
Out[146]:

```
array([[[1., 0.],
        [0., 1.]],

       [[0., 1.],
        [1., 0.]],

       [[0., 1.],
        [1., 0.]],

       ...,

       [[0., 1.],
        [1., 0.]],

       [[0., 1.],
        [1., 0.]],

       [[1., 0.],
        [0., 1.]]], dtype=float32)
```

In [147]:

```
import tensorflow as tf
tf.keras.utils.plot_model(model, show_shapes=True)
```

Out[147]:



In [148]:

```
print(model.summary())

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='Adam',
               metrics=['accuracy', Precision(), Recall()])

# Train model

batch_size = 64
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_9 (Embedding)      (None, 50, 1100)          11000000  
_________________________________________________________________
lstm_10 (LSTM)               (None, 50, 64)            298240    
```

```
_________________________________________________________________
dropout_10 (Dropout)         (None, 50, 64)            0         
_________________________________________________________________
lstm_11 (LSTM)               (None, 8)                 2336      
_________________________________________________________________
dropout_11 (Dropout)         (None, 8)                 0         
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 18        
=================================================================
Total params: 11,300,594
Trainable params: 11,300,594
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
132/132 [==============================] - 49s 349ms/step - loss: 1.2326 - accuracy: 0.61
31 - precision_9: 0.6131 - recall_9: 0.6131 - val_loss: 0.6603 - val_accuracy: 0.6222 - v
al_precision_9: 0.6222 - val_recall_9: 0.6222
Epoch 2/10
132/132 [==============================] - 46s 347ms/step - loss: 0.6835 - accuracy: 0.61
31 - precision_9: 0.6131 - recall_9: 0.6131 - val_loss: 0.6478 - val_accuracy: 0.6222 - v
al_precision_9: 0.6222 - val_recall_9: 0.6222
Epoch 3/10
132/132 [==============================] - 46s 346ms/step - loss: 0.4213 - accuracy: 0.78
61 - precision_9: 0.7861 - recall_9: 0.7861 - val_loss: 0.2831 - val_accuracy: 0.8913 - v
al_precision_9: 0.8913 - val_recall_9: 0.8913
Epoch 4/10
132/132 [==============================] - 46s 346ms/step - loss: 0.2471 - accuracy: 0.92
30 - precision_9: 0.9230 - recall_9: 0.9230 - val_loss: 0.2721 - val_accuracy: 0.9051 - v
al_precision_9: 0.9051 - val_recall_9: 0.9051
Epoch 5/10
132/132 [==============================] - 46s 346ms/step - loss: 0.1844 - accuracy: 0.95
08 - precision_9: 0.9508 - recall_9: 0.9508 - val_loss: 0.2795 - val_accuracy: 0.8989 - v
al_precision_9: 0.8989 - val_recall_9: 0.8989
Epoch 6/10
132/132 [==============================] - 46s 348ms/step - loss: 0.1394 - accuracy: 0.96
39 - precision_9: 0.9639 - recall_9: 0.9639 - val_loss: 0.3264 - val_accuracy: 0.9070 - v
al_precision_9: 0.9070 - val_recall_9: 0.9070
Epoch 7/10
132/132 [==============================] - 46s 349ms/step - loss: 0.1212 - accuracy: 0.97
13 - precision_9: 0.9713 - recall_9: 0.9713 - val_loss: 0.3455 - val_accuracy: 0.8994 - v
al_precision_9: 0.8994 - val_recall_9: 0.8994
Epoch 8/10
132/132 [==============================] - 46s 347ms/step - loss: 0.1003 - accuracy: 0.97
70 - precision_9: 0.9770 - recall_9: 0.9770 - val_loss: 0.3797 - val_accuracy: 0.9032 - v
al_precision_9: 0.9032 - val_recall_9: 0.9032
Epoch 9/10
132/132 [==============================] - 46s 347ms/step - loss: 0.0859 - accuracy: 0.98
14 - precision_9: 0.9814 - recall_9: 0.9814 - val_loss: 0.3402 - val_accuracy: 0.8956 - v
al_precision_9: 0.8956 - val_recall_9: 0.8956
Epoch 10/10
132/132 [==============================] - 46s 345ms/step - loss: 0.0699 - accuracy: 0.98
42 - precision_9: 0.9842 - recall_9: 0.9842 - val_loss: 0.5881 - val_accuracy: 0.8942 - v
al_precision_9: 0.8942 - val_recall_9: 0.8942
```

Model Accuracy & Loss

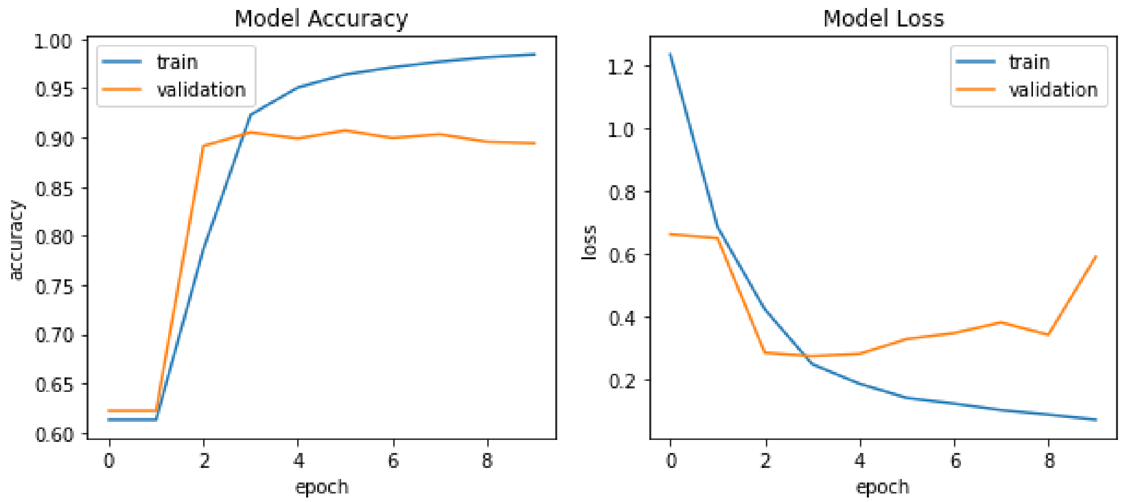
In [149]:

```
# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test, verbose=0)
# Print metrics
print('')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))
```

```
Accuracy  : 0.9104
Precision : 0.9104
Recall    : 0.9104
F1 Score  : 0.9104
```

In [150]:

```
def plot_training_hist(history):
    '''Function to plot history for accuracy and loss'''

    fig, ax = plt.subplots(1, 2, figsize=(10,4))
    # first plot
    ax[0].plot(history.history['accuracy'])
    ax[0].plot(history.history['val_accuracy'])
    ax[0].set_title('Model Accuracy')
    ax[0].set_xlabel('epoch')
    ax[0].set_ylabel('accuracy')
    ax[0].legend(['train', 'validation'], loc='best')
    # second plot
    ax[1].plot(history.history['loss'])
    ax[1].plot(history.history['val_loss'])
    ax[1].set_title('Model Loss')
    ax[1].set_xlabel('epoch')
    ax[1].set_ylabel('loss')
    ax[1].legend(['train', 'validation'], loc='best')

plot_training_hist(history)
```



In [151]:

```
# Save the model architecture & the weights
model.save('best_model.h5')
print('Best model saved')
```

```
Best model saved
```

In [152]:

```
from keras.models import load_model

# Load model
model = load_model('best_model.h5')

def predict_class(text):
    '''Function to predict sentiment class of the passed text'''

    sentiment_classes = ['tidak kasar', 'kasar']
    max_len=50

    # Transforms text to a sequence of integers using a tokenizer object
    xt = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    xt = pad_sequences(xt, padding='post', maxlen=max_len)
    # Do the prediction using the loaded model
    yt = model.predict(xt).argmax(axis=1)
    # Print the predicted sentiment
    print(yt)
    print('Kata Tersebut mengandung konotasi', sentiment_classes[yt[0]])
```

Data Prediction 1

In [153]:

```
predict_class(['halo semuanya aku baik kan'])
```

```
[0]
Kata Tersebut mengandung konotasi tidak kasar
```

Data Prediction 2

In [154]:

```
predict_class(['apaan sih lo anjing banget dah'])
```

```
[1]
Kata Tersebut mengandung konotasi kasar
```

70 30 ratio

Performing learning for 70% data training and 30% data testing.

Train & Test Split

In [155]:

```
y = pd.get_dummies(df['Abusive'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.3, rando
m_state=1)
print('Train Set ->', X_train.shape, y_train.shape)
print('Validation Set ->', X_val.shape, y_val.shape)
print('Test Set ->', X_test.shape, y_test.shape)
```

```
Train Set -> (6452, 50) (6452, 2)
Validation Set -> (2766, 50) (2766, 2)
Test Set -> (3951, 50) (3951, 2)
```

In [156]:

```
import keras.backend as K

def f1_score(precision, recall):
    ''' Function to calculate f1 score '''

    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

Bidirectional LSTM Using NN

In [157]:

```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dr
opout
from keras.metrics import Precision, Recall
from keras import datasets

from keras.callbacks import LearningRateScheduler
from keras.callbacks import History

from keras import losses

vocab_size = 10000
embedding_size = 1100
epochs=10
```

```
# Build model
model= Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model.add(LSTM(64, activation="ReLU", return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(8, activation="ReLU"))
model.add(Dropout(0.2))
model.add(Dense(2, activation='softmax'))
```

In [158]:

```
import tensorflow
tensorflow.keras.utils.to_categorical(y_train)
tensorflow.keras.utils.to_categorical(y_test)
```

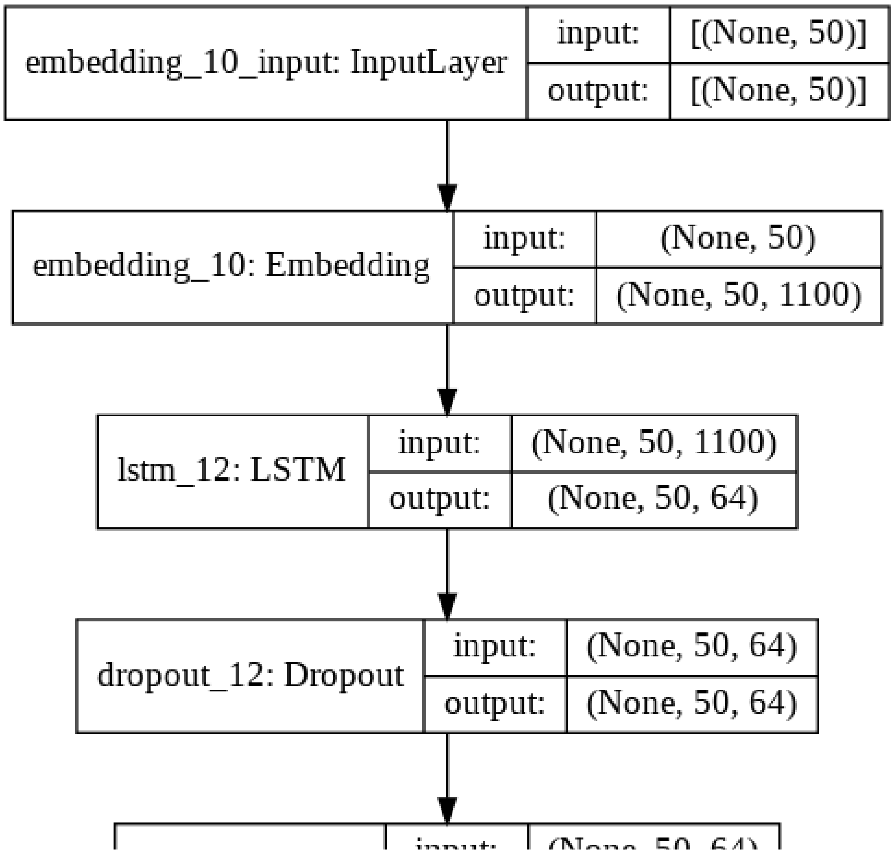
Out[158]:

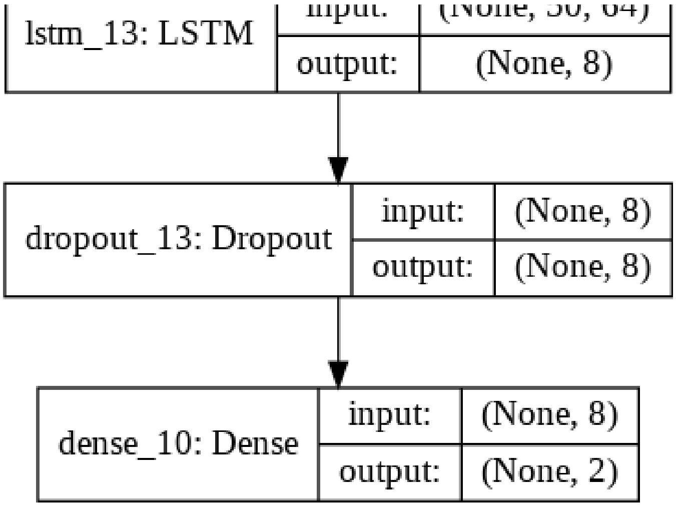
```
array([[[1., 0.],
        [0., 1.]],

       [[0., 1.],
        [1., 0.]],

       [[0., 1.],
        [1., 0.]],

       ...,

       [[1., 0.],
        [0., 1.]],

       [[0., 1.],
        [1., 0.]],

       [[1., 0.],
        [0., 1.]]], dtype=float32)
```

In [159]:

```
import tensorflow as tf
tf.keras.utils.plot_model(model, show_shapes=True)
```

Out[159]:





In [160]:

```
print(model.summary())

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
               metrics=['accuracy', Precision(), Recall()])

# Train model

batch_size = 64
history = model.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_10 (Embedding)     (None, 50, 1100)          11000000  
_________________________________________________________________
lstm_12 (LSTM)               (None, 50, 64)            298240    
_________________________________________________________________
dropout_12 (Dropout)         (None, 50, 64)            0         
_________________________________________________________________
lstm_13 (LSTM)               (None, 8)                 2336      
_________________________________________________________________
dropout_13 (Dropout)         (None, 8)                 0         
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 18        
=================================================================
Total params: 11,300,594
Trainable params: 11,300,594
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
101/101 [==============================] - 40s 371ms/step - loss: 0.6694 - accuracy: 0.61
48 - precision_10: 0.6148 - recall_10: 0.6148 - val_loss: 0.6686 - val_accuracy: 0.6095 -
val_precision_10: 0.6095 - val_recall_10: 0.6095
Epoch 2/10
101/101 [==============================] - 37s 363ms/step - loss: 1.1402 - accuracy: 0.61
79 - precision_10: 0.6179 - recall_10: 0.6179 - val_loss: 0.5666 - val_accuracy: 0.6092 -
val_precision_10: 0.6092 - val_recall_10: 0.6092
Epoch 3/10
101/101 [==============================] - 37s 363ms/step - loss: 0.5365 - accuracy: 0.75
65 - precision_10: 0.7565 - recall_10: 0.7565 - val_loss: 0.4762 - val_accuracy: 0.8453 -
val_precision_10: 0.8453 - val_recall_10: 0.8453
Epoch 4/10
101/101 [==============================] - 37s 362ms/step - loss: 0.4707 - accuracy: 0.81
12 - precision_10: 0.8112 - recall_10: 0.8112 - val_loss: 0.3274 - val_accuracy: 0.8803 -
val_precision_10: 0.8803 - val_recall_10: 0.8803
Epoch 5/10
101/101 [==============================] - 36s 361ms/step - loss: 0.2557 - accuracy: 0.90
78 - precision_10: 0.9078 - recall_10: 0.9078 - val_loss: 0.3053 - val_accuracy: 0.8832 -
```

```
70 - precision_10: 0.9070 - recall_10: 0.9070 - val_loss: 0.3055 - val_accuracy: 0.8832 -
val_precision_10: 0.8832 - val_recall_10: 0.8832
Epoch 6/10
101/101 [==============================] - 37s 363ms/step - loss: 0.1972 - accuracy: 0.92
39 - precision_10: 0.9239 - recall_10: 0.9239 - val_loss: 0.3347 - val_accuracy: 0.8839 -
val_precision_10: 0.8839 - val_recall_10: 0.8839
Epoch 7/10
101/101 [==============================] - 36s 359ms/step - loss: 0.1808 - accuracy: 0.93
99 - precision_10: 0.9399 - recall_10: 0.9399 - val_loss: 0.3675 - val_accuracy: 0.8847 -
val_precision_10: 0.8847 - val_recall_10: 0.8847
Epoch 8/10
101/101 [==============================] - 36s 360ms/step - loss: 0.1420 - accuracy: 0.95
07 - precision_10: 0.9507 - recall_10: 0.9507 - val_loss: 0.4101 - val_accuracy: 0.8868 -
val_precision_10: 0.8868 - val_recall_10: 0.8868
Epoch 9/10
101/101 [==============================] - 36s 360ms/step - loss: 0.1374 - accuracy: 0.95
60 - precision_10: 0.9560 - recall_10: 0.9560 - val_loss: 0.4157 - val_accuracy: 0.8825 -
val_precision_10: 0.8825 - val_recall_10: 0.8825
Epoch 10/10
101/101 [==============================] - 36s 359ms/step - loss: 0.1120 - accuracy: 0.95
95 - precision_10: 0.9595 - recall_10: 0.9595 - val_loss: 0.5013 - val_accuracy: 0.8738 -
val_precision_10: 0.8738 - val_recall_10: 0.8738
```

Model Accuracy & Loss

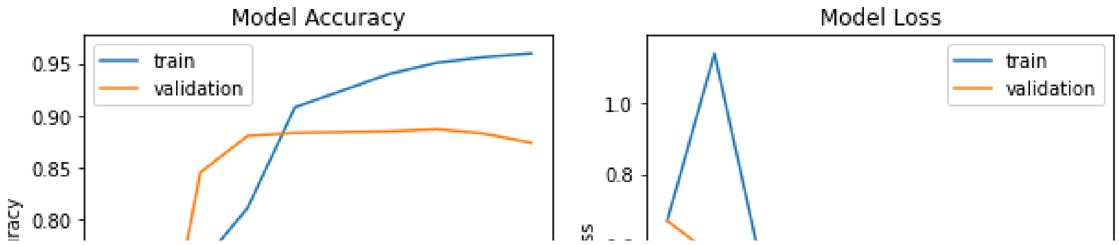
In [161]:

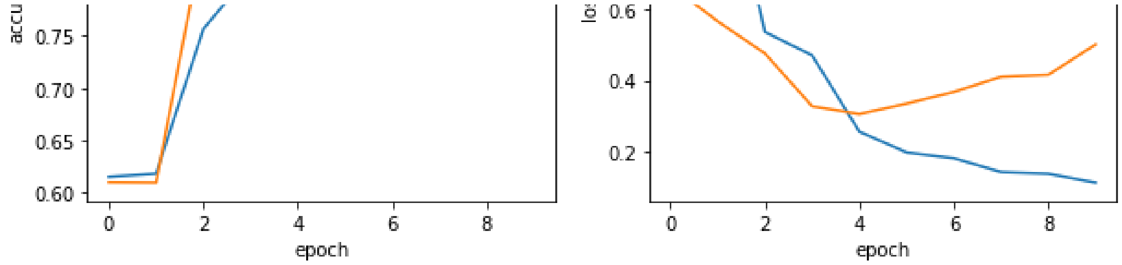
```
# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test, verbose=0)
# Print metrics
print('')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))
```

```
Accuracy  : 0.8924
Precision : 0.8924
Recall    : 0.8924
F1 Score  : 0.8924
```

In [162]:

```
def plot_training_hist(history):
    '''Function to plot history for accuracy and loss'''

    fig, ax = plt.subplots(1, 2, figsize=(10,4))
    # first plot
    ax[0].plot(history.history['accuracy'])
    ax[0].plot(history.history['val_accuracy'])
    ax[0].set_title('Model Accuracy')
    ax[0].set_xlabel('epoch')
    ax[0].set_ylabel('accuracy')
    ax[0].legend(['train', 'validation'], loc='best')
    # second plot
    ax[1].plot(history.history['loss'])
    ax[1].plot(history.history['val_loss'])
    ax[1].set_title('Model Loss')
    ax[1].set_xlabel('epoch')
    ax[1].set_ylabel('loss')
    ax[1].legend(['train', 'validation'], loc='best')

plot_training_hist(history)
```





In [163]:

```
# Save the model architecture & the weights
model.save('best_model.h5')
print('Best model saved')
```

```
Best model saved
```

In [164]:

```
from keras.models import load_model

# Load model
model = load_model('best_model.h5')

def predict_class(text):
    '''Function to predict sentiment class of the passed text'''

    sentiment_classes = ['tidak kasar', 'kasar']
    max_len=50

    # Transforms text to a sequence of integers using a tokenizer object
    xt = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    xt = pad_sequences(xt, padding='post', maxlen=max_len)
    # Do the prediction using the loaded model
    yt = model.predict(xt).argmax(axis=1)
    # Print the predicted sentiment
    print(yt)
    print('Kata Tersebut mengandung konotasi', sentiment_classes[yt[0]])
```

Data Prediction 1

In [165]:

```
predict_class(['halo semuanya aku baik kan'])
```

```
[0]
Kata Tersebut mengandung konotasi tidak kasar
```

Data Prediction 2

In [166]:

```
predict_class(['apaan sih lo anjing banget dah'])
```

```
[1]
Kata Tersebut mengandung konotasi kasar
```

60 40 ratio

Performing learning for 60% data training and 40% data testing.

Train & Test Split

In [167]:

```
y = pd.get_dummies(df['Abusive'])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.4, random_state=1)
print('Train Set ->', X_train.shape, y_train.shape)
print('Validation Set ->', X_val.shape, y_val.shape)
print('Test Set ->', X_test.shape, y_test.shape)
```

```
Train Set -> (4740, 50) (4740, 2)
Validation Set -> (3161, 50) (3161, 2)
Test Set -> (5268, 50) (5268, 2)
```

In [168]:

```
import keras.backend as K

def f1_score(precision, recall):
    ''' Function to calculate f1 score '''

    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

Bidirectional LSTM Using NN

In [169]:

```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dropout
from keras.metrics import Precision, Recall
from keras import datasets

from keras.callbacks import LearningRateScheduler
from keras.callbacks import History

from keras import losses

vocab_size = 10000
embedding_size = 1100
epochs=10

# Build model
model= Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model.add(LSTM(64, activation="ReLU", return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(8, activation="ReLU"))
model.add(Dropout(0.2))
model.add(Dense(2, activation='softmax'))
```

In [170]:

```
import tensorflow
tensorflow.keras.utils.to_categorical(y_train)
tensorflow.keras.utils.to_categorical(y_test)
```

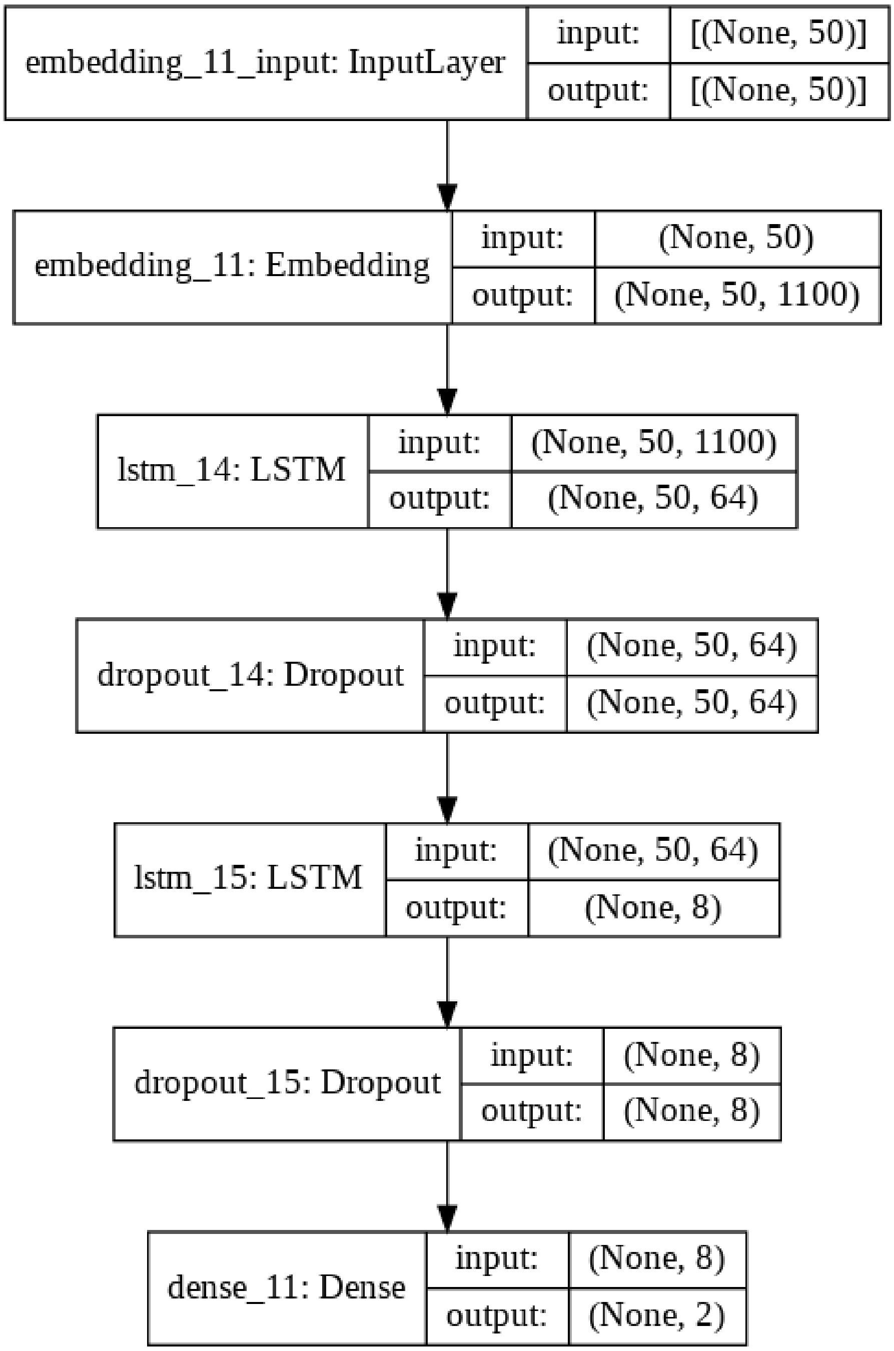
Out[170]:

```
array([[[1., 0.],
        [0., 1.]],

       [[0., 1.],
        [1., 0.]],

       [[0., 1.],
        [1., 0.]],

       ...,

       [[0., 1.],
        [1., 0.]],

       [[0., 1.],
       [[0., 1.],
        [1., 0.]],

       [[0., 1.],
        [1., 0.]]], dtype=float32)
```

In [171]:

```
import tensorflow as tf
tf.keras.utils.plot_model(model, show_shapes=True)
```

Out[171]:



In [172]:

```
print(model.summary())

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy', Precision(), Recall()])

# Train model

batch_size = 64
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_11 (Embedding)     (None, 50, 1100)          11000000  
_________________________________________________________________
lstm_14 (LSTM)               (None, 50, 64)            298240    
_________________________________________________________________
dropout_14 (Dropout)         (None, 50, 64)            0         
_________________________________________________________________
lstm_15 (LSTM)               (None, 8)                 2336      
_________________________________________________________________
dropout_15 (Dropout)         (None, 8)                 0         
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 18        
=================================================================
Total params: 11,300,594
Trainable params: 11,300,594
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
75/75 [==============================] - 33s 410ms/step - loss: 0.6744 - accuracy: 0.6097
- precision_11: 0.6097 - recall_11: 0.6097 - val_loss: 0.6697 - val_accuracy: 0.6125 - va
l_precision_11: 0.6125 - val_recall_11: 0.6125
Epoch 2/10
75/75 [==============================] - 28s 378ms/step - loss: 0.6273 - accuracy: 0.6437
- precision_11: 0.6437 - recall_11: 0.6437 - val_loss: 0.4801 - val_accuracy: 0.7760 - va
l_precision_11: 0.7760 - val_recall_11: 0.7760
Epoch 3/10
75/75 [==============================] - 29s 381ms/step - loss: 0.7521 - accuracy: 0.8162
- precision_11: 0.8162 - recall_11: 0.8162 - val_loss: 0.2954 - val_accuracy: 0.8804 - va
l_precision_11: 0.8804 - val_recall_11: 0.8804
Epoch 4/10
75/75 [==============================] - 29s 383ms/step - loss: 0.2447 - accuracy: 0.8854
- precision_11: 0.8854 - recall_11: 0.8854 - val_loss: 0.2972 - val_accuracy: 0.8858 - va
l_precision_11: 0.8858 - val_recall_11: 0.8858
Epoch 5/10
75/75 [==============================] - 29s 384ms/step - loss: 0.1704 - accuracy: 0.9416
- precision_11: 0.9416 - recall_11: 0.9416 - val_loss: 0.3116 - val_accuracy: 0.8880 - va
l_precision_11: 0.8880 - val_recall_11: 0.8880
Epoch 6/10
75/75 [==============================] - 29s 381ms/step - loss: 0.1083 - accuracy: 0.9561
- precision_11: 0.9561 - recall_11: 0.9561 - val_loss: 0.3621 - val_accuracy: 0.8905 - va
l_precision_11: 0.8905 - val_recall_11: 0.8905
Epoch 7/10
75/75 [==============================] - 29s 385ms/step - loss: 0.0801 - accuracy: 0.9582
- precision_11: 0.9582 - recall_11: 0.9582 - val_loss: 0.6985 - val_accuracy: 0.8842 - va
l_precision_11: 0.8842 - val_recall_11: 0.8842
Epoch 8/10
75/75 [==============================] - 29s 383ms/step - loss: 0.0566 - accuracy: 0.9696
- precision_11: 0.9696 - recall_11: 0.9696 - val_loss: 0.7819 - val_accuracy: 0.8855 - va
l_precision_11: 0.8855 - val_recall_11: 0.8855
Epoch 9/10
75/75 [==============================] - 29s 382ms/step - loss: 0.0490 - accuracy: 0.9776
- precision_11: 0.9776 - recall_11: 0.9776 - val_loss: 0.5963 - val_accuracy: 0.8836 - va
l_precision_11: 0.8836 - val_recall_11: 0.8836
Epoch 10/10
75/75 [==============================] - 29s 382ms/step - loss: 0.0412 - accuracy: 0.9844
- precision_11: 0.9844 - recall_11: 0.9844 - val_loss: 1.0114 - val_accuracy: 0.8867 - va
l_precision_11: 0.8867 - val_recall_11: 0.8867
```

Model Accuracy & Loss

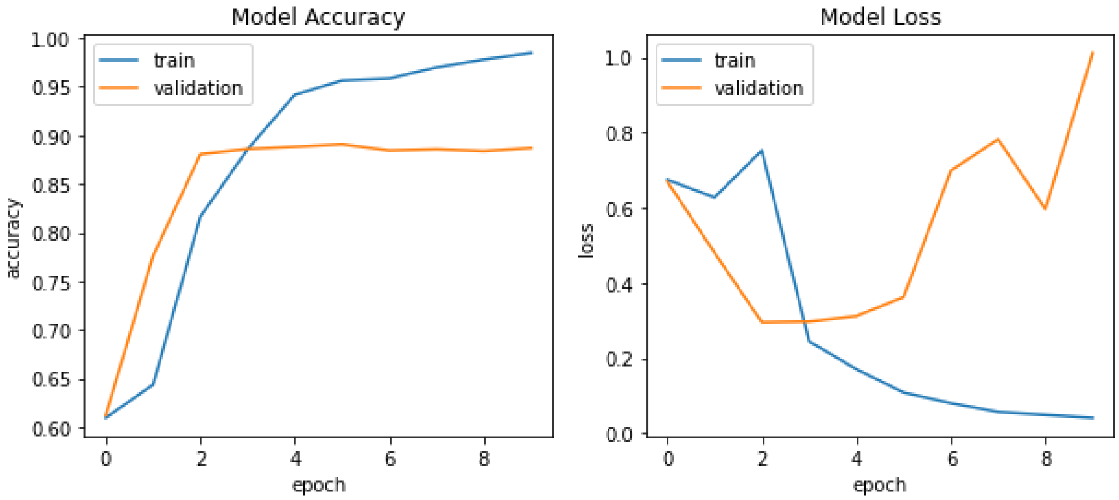
In [173]:

```
# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test, verbose=0)
# Print metrics
print('')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))
```

```
Accuracy  : 0.8724
Precision : 0.8724
Recall    : 0.8724
F1 Score  : 0.8724
```

In [174]:

```
def plot_training_hist(history):
    '''Function to plot history for accuracy and loss'''

    fig, ax = plt.subplots(1, 2, figsize=(10,4))
    # first plot
    ax[0].plot(history.history['accuracy'])
    ax[0].plot(history.history['val_accuracy'])
    ax[0].set_title('Model Accuracy')
    ax[0].set_xlabel('epoch')
    ax[0].set_ylabel('accuracy')
    ax[0].legend(['train', 'validation'], loc='best')
    # second plot
    ax[1].plot(history.history['loss'])
    ax[1].plot(history.history['val_loss'])
    ax[1].set_title('Model Loss')
    ax[1].set_xlabel('epoch')
    ax[1].set_ylabel('loss')
    ax[1].legend(['train', 'validation'], loc='best')

plot_training_hist(history)
```



In [175]:

```
# Save the model architecture & the weights
model.save('best_model.h5')
print('Best model saved')
```

```
Best model saved
```

In [176]:

```
from keras.models import load_model

# Load model
model = load_model('best_model.h5')

def predict_class(text):
    '''Function to predict sentiment class of the passed text'''

    sentiment_classes = ['tidak kasar', 'kasar']
    max_len=50
    # Transforms text to a sequence of integers using a tokenizer object
    xt = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    xt = pad_sequences(xt, padding='post', maxlen=max_len)
    # Do the prediction using the loaded model
    yt = model.predict(xt).argmax(axis=1)
    # Print the predicted sentiment
    print(yt)
    print('Kata Tersebut mengandung konotasi', sentiment_classes[yt[0]])
```

Data Prediction 1

In [177]:

```
predict_class(['halo semuanya aku baik kan'])
```

```
[0]
Kata Tersebut mengandung konotasi tidak kasar
```

Data Prediction 2

In [178]:

```
predict_class(['apaan sih lo anjing banget dah'])
```

```
[1]
Kata Tersebut mengandung konotasi kasar
```