Genome Sequence Classification Using Long Short-Term Memory Recurrent Neural Network

Siamak Rabienia

The Data Incubator

A Pragmatic Institute Company

Spring 2020

Problem Definition

Objective: Given sequence of genes, train a deep neural network for pattern recognition.

“….TAAATGCTACCGTCACAATATTCGTG
ACCAGACTGAAGTTATTGAATGCGGCTT
GGAAGTTAAGCACTTATGCCC….”

“….TAAATGCTACCGTCACAATATTCGTG
ACCAGACTGAAGTTATTGAATGCGGCTT
GGAAGTTAAGCACTTATGCCC….”

Dataset.npy

- Consists of genes and their resistances status against a drug.
- A gene is a string of DNA letters that gets translated into a protein, or amino acid.
- Size of genome sequences are variable, therefore we perform padding to fix size.

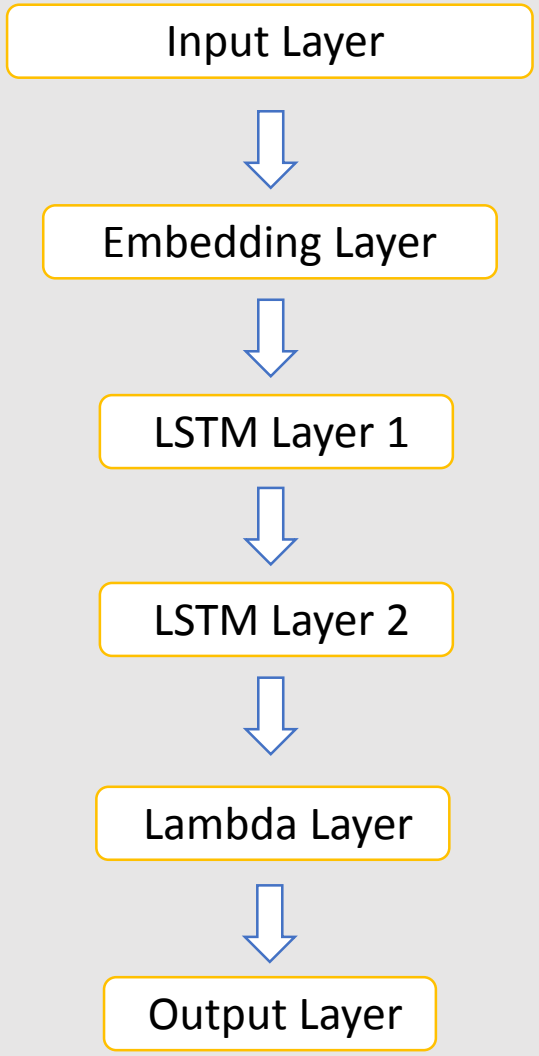
Application: Find treatment for diseases

Criteria:

- Use 5-fold cross-validation to select an optimal hyper parameter set.
- Explain your choice of model architecture and any challenges that you encountered.
- Compare CV performance to test performance.

Metric: Accuracy

Structure LSTM-RNN



Input Layer: Genome Sequence of size 190 (X)

Embedding Layer: Projects the input space to the lower dimension spaces

LSTM Layers: A long short-term memory (LSTM) network that avoids the vanishing gradient problem by adding 'forget' gates

Lambda Layers: Wraps an arbitrary expression.

Output Layer: $Z_j^{[i]} = W_j^{[i]^T} X + B_j^{[i]}$

W: weight, B: bias, Z: output

Analysis

Activation Function: **Sigmoid function** is used to activate the neurons

$$g(z) = \frac{1}{1+e^{-z}}$$

It converts estimated probabilities to 0 and 1.

Cross-Entropy Loss:

$$L(z,y) = -[y \log(z) + (1 - y)\log(1 - z)]$$

Updating Weights:

- Take a batch of training data.
- Perform forward propagation to obtain the loss.
- Backpropagate the loss to get the gradients.
- Use the gradients to update the network's weights.

Learning Rate: It will be adapted using Adam's optimizer.

Dropout: **18% Portion** - Acts like a regularizer preventing overfitting the training data by dropping out units in a neural network.

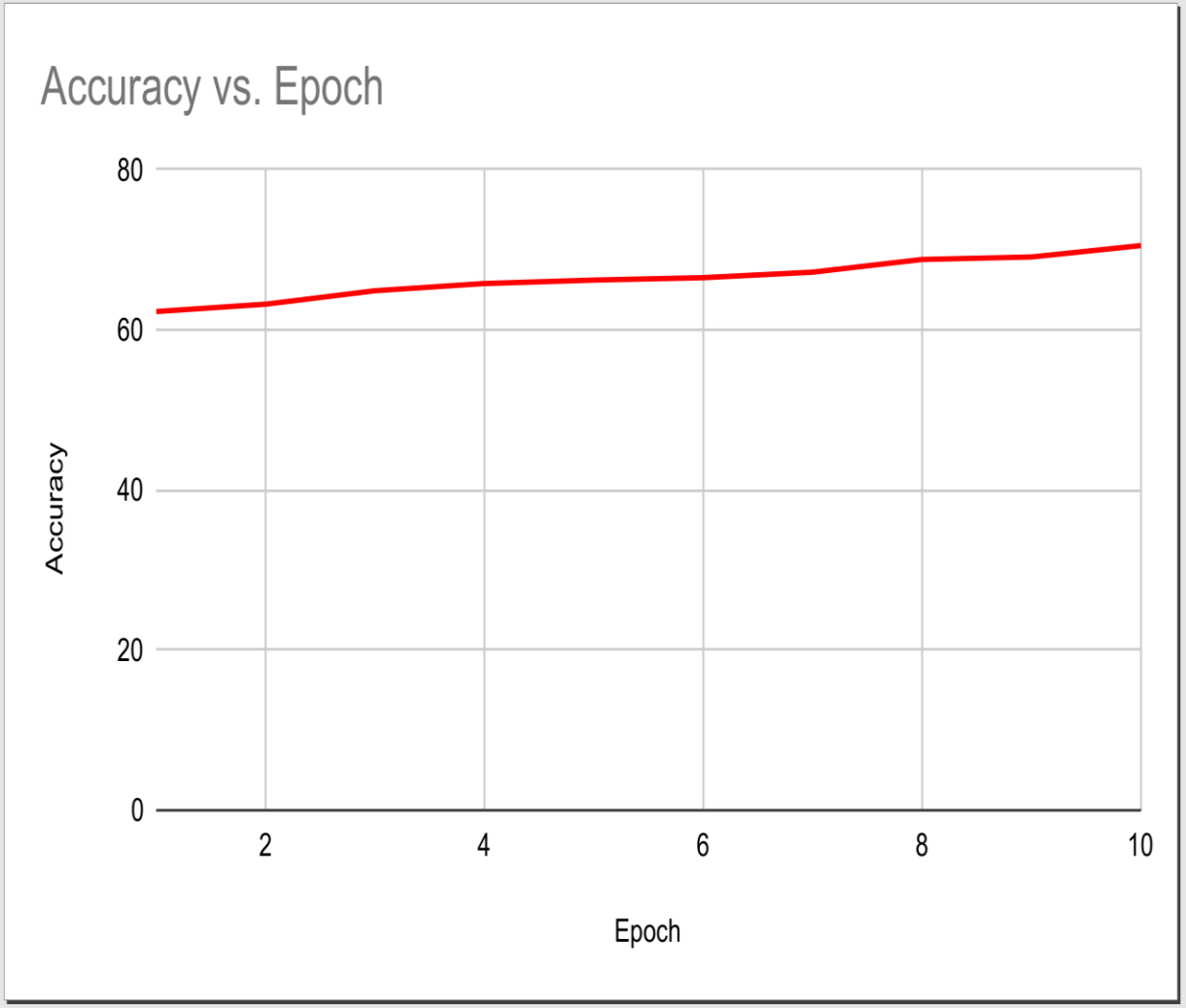
Train-Test: **80%-20%** with 5-fold cross-validation to avoid over-fitting issues and discover the hyper-parameters (epoch size, neuron size)

Epoch: **10**

Neuron size: **60**

Batch Size: **400**

Result


Accuracy vs. Epoch
80
60
40
20
0
Accuracy
2
4
6
8
10
Epoch


Train Loss and Test Loss
Train Loss
Test Loss
1.25
1
0.75
0.5
0.25
0
2
4
6
8
10
Epoch

Python_Code

```python
import tensorflow as tf
import theano
import pandas as pd
import numpy as np
import matplotlib
import os
import math
import pydot
import graphviz
matplotlib.use('pdf')
import matplotlib.pyplot as plt
from keras.layers import Dense, Dropout, LSTM, Embedding, Activation, Lambda, Bidirectional
from keras.engine import Input, Model, InputSpec
from keras.preprocessing.sequence import pad_sequences
from keras.utils import plot_model
from keras.utils.data_utils import get_file
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from sklearn.utils import class_weight
from keras import backend as K
from keras.preprocessing import sequence
from keras.models import model_from_json


EPCOHS = 10
BATCH_SIZE = 400
input_layer = 4
output_layer = 50
middle_layer = 62
dropout_perentage = 0.18
seqlen = 900
checkpoint_dir ='checkpoints'
os.path.exists(checkpoint_dir)

def letter_to_index(letter):
    _alphabet = 'ATGC'
    return next((i for i, _letter in enumerate(_alphabet) if _letter == letter), None)

def parseInput(test_split = 0.1, maxlen = seqlen):
    data = np.load("dataset.npy", allow_pickle=True).item()
    df = pd.DataFrame(data, columns=["genes", "resistant"])
    df['genes'] = df['genes'].apply(lambda x: [int(letter_to_index(e)) for e in x])
    df = df.reindex(np.random.permutation(df.index))
    train_size = int(len(df) * (1 - test_split))
    X_train = df['genes'].values[:train_size]
    y_train = np.array(df['resistant'].values[:train_size])
    X_test = np.array(df['genes'].values[train_size:])
    y_test = np.array(df['resistant'].values[train_size:])
    return pad_sequences(X_train, maxlen=maxlen), y_train, pad_sequences(X_test, maxlen=maxlen), y_test
```

```python
def build_net(input_length, rnn_hidden_dim = middle_layer, output_dim = output_layer, input_dim = input_layer, dropout = dropout_
    model = Sequential()
    model.add(Embedding(input_dim = input_layer, output_dim = output_dim, input_length = input_length, name='embedding_layer'))
    model.add(Bidirectional(LSTM(rnn_hidden_dim, return_sequences=True)))
    model.add(Dropout(dropout))
    model.add(Bidirectional(LSTM(rnn_hidden_dim)))
    model.add(Dropout(dropout))
    model.add(Dense(1, activation='sigmoid'))
    model.compile('adam', 'binary_crossentropy', metrics=['accuracy'])
    return model

def create_plots(history):
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('accuracy.png')
    plt.clf()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('loss.png')
    plt.clf()

if __name__ == '__main__':
    X_train, y_train, X_test, y_test = parseInput()
    model = build_net(len(X_train[0]))

    filepath= checkpoint_dir + "/weights-improvement-{epoch:02d}-{val_loss:.2f}.hdf5"
    checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=0, save_best_only=True, mode='max')
    callbacks_list = [checkpoint]

    class_weight = class_weight.compute_class_weight('balanced', np.unique(y_train), y_train)
    print(class_weight)
    history = model.fit(X_train, y_train, batch_size=BATCH_SIZE, class_weight=class_weight,
        epochs=EPCOHS, callbacks=callbacks_list, validation_split = 0.1, verbose = 1)

    model_json = model.to_json()
    with open("neuralnet.json", "w") as json_file:
        json_file.write(model_json)

    model.save_weights("weights")
    create_plots(history)
    plot_model(model, to_file='model.png')

    score, acc = model.evaluate(X_test, y_test, batch_size=BATCH_SIZE)
    print('score:', score)
    print(' accuracy:', acc)
```